A Modular Quantum Network Architecture for Integrating Network Scheduling with Local Program Execution

Beauchamp, Thomas R.; Jirovská, Hana; Gauthier, Scarlett; Wehner, Stephanie

DOI
10.1109/TQE.2025.3624658

Publication date
2025

Document Version
Final published version

Published in
IEEE Transactions on Quantum Engineering

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/TQE.2020.DOI

A Modular Quantum Network Architecture for Integrating Network Scheduling with Local Program Execution.

THOMAS R. BEAUCHAMP[1,2,3], HANA JIROVSKÁ[1,2,3], SCARLETT GAUTHIER [1,2,3] and STEPHANIE WEHNER[1,2,3]

[1]QuTech, Delft University of Technology, Lorentzweg 1, 2628CJ Delft, The Netherlands
[2]Kavli Institute of Nanoscience, Delft University of Technology, Lorentzweg 1, 2628 CJ Delft, The Netherlands
[3]Quantum Computer Science, EEMCS, Delft University of Technology, Lorentzweg 1, 2628 CJ Delft, The Netherlands

Corresponding author: Thomas Beauchamp (email: t.r.beauchamp@tudelft.nl).

This work was supported by the Quantum Internet Alliance (QIA). QIA has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No. 101102140. SW also acknowledges funding from NWO VICI. This work is an extended version of a conference paper "*Extended Abstract: A Modular Quantum Network Architecture for Integrating Network Scheduling with Local Program Execution.*", IEEE INFOCOM 2025 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), (IN PRESS)

ABSTRACT We propose an architecture for scheduling network operations enabling the end-to-end generation of entanglement according to user demand. The main challenge solved by this architecture is to allow for the integration of a network schedule with the execution of quantum programs running on processing end nodes in order to realise quantum network applications. A key element of this architecture is the definition of an entanglement packet to meet application requirements on near-term quantum networks where the lifetimes of the qubits stored at the end nodes are limited. Our architecture is fully modular and hardware agnostic, and defines a framework for further research on specific components that can now be developed independently of each other. In order to evaluate our architecture, we realise a proof of concept implementation on a simulated 6-node network in a star topology. We show our architecture facilitates the execution of quantum network applications, and that robust admission control is required to maintain quality of service. Finally, we comment on potential bottlenecks in our architecture and provide suggestions for future improvements.

INDEX TERMS Quantum networks, network architecture, network scheduling

I. INTRODUCTION

The realisation of a quantum internet will enable the use of new networked applications beyond what is possible with the current classical internet. Such applications include the ability to perform verifiably secure secret sharing [1], [2], secure remote computation [3]–[5] and securely electing a leader [6], amongst many others [7]. The aim of any quantum network architecture therefore should be to ensure that these applications can be successfully executed.

To execute a quantum application, so-called *entangled links* between the *end nodes*, the quantum devices the users have access to, are required. Each of these entangled links is a pair of entangled qubits with a fidelity with respect to an EPR pair [8], where the fidelity is a measure of the quality of the entangled link [9]. However, such links are difficult to produce and doing so requires the use of a limited quantity of resources in the quantum network [10]–[13]. Furthermore, at present each link has a limited usable lifetime as quantum memories experience decoherence over time, reducing the quality of stored entangled links [14]. Therefore, there exist two interacting scheduling problems which must be solved: firstly how should the limited network resources be assigned to pairs of users to allow them to generate entangled links (the *network scheduling problem*),

and secondly how to efficiently schedule the execution of quantum applications to efficiently use any entangled links which are generated (the *local scheduling problem*).

The first operating system for end nodes, QNodeOS [15], allows the second of these problems to be solved. *Qoala* [16], an application execution environment which runs on QNodeOS, offers an improved end node execution environment. QNodeOS breaks an application into a program at each node, which is then further subdivided into blocks of instructions that can be scheduled for execution at runtime. Furthermore, Qoala enables a compiler which can provide advice about the quantity and quality of entangled links which are required. Although Qoala successfully address the local scheduling problem, it requires the existence of a network schedule to supply allocations of time during which entanglement generation can take place.

There currently does not exist a network architecture which can produce schedules which are compatible with such an execution environment. Without such an architecture, the network scheduling problem cannot be solved in a manner which still allows the local scheduling problem to be solved effectively. In this work we therefore propose such an architecture to unify the approach to the network and local scheduling problems. In particular:

- **We introduce the first quantum network architecture which takes a unified approach to scheduling the execution of quantum applications on end nodes and scheduling the use of resources on the network**. As part of this we introduce a notion of **packets of entanglement** to capture the requirements on entangled links imposed by applications, and a corresponding notion of a **packet generation task** to allow the network to efficiently schedule time for the generation of these packets of entanglement. We also define a **demand format** which captures all the information required by the central controller to be able to compute network schedules. This architecture also provides a **modular framework** within which further research can be undertaken to develop synergistic network and local scheduling strategies. This will enable network schedules to be integrated into local program execution in a well-defined and consistent manner.
- **We provide an example implementation of our architecture in simulation**, using earliest deadline first derived methods for network scheduling. We use this implementation to perform numerical simulations of our architecture and create a baseline performance evaluation against which to benchmark further work in the domain of quantum network scheduling. The code used for this implementation and the data used in the evaluation is available from [17], [18].

A. STRUCTURE

The rest of this paper is structured as follows: In II we give some background and related work and discuss how our work here differs. In III we discuss design considerations which inform the design of our architecture. In IV we lay out our proposed network architecture. In V we give an example of an implementation of the architecture. Finally in VI we evaluate the performance of our architecture using a specific implementation, and in VII we comment on future directions for research. A table summarising all the notation we use in this paper can be found in Appendix A.

II. BACKGROUND

A. NETWORK MODEL

There are many different models for how a quantum network should operate. In the literature, one common example is what we will call a *pre-loaded* network, where there is a high probability that entangled links are immediately available to an application (e.g. [19]–[22]). Such networks rely on continuously generating and buffering entanglement between each pair of network components. However, restrictions on achievable entanglement generation rates, buffer lifetimes and buffer capacities limit the possibility of near-term implementations of such networks. For example, experiments on leading hardware [23] have realised a three-node network, on which they reported memory lifetimes of 11ms and generation rates of one end-to-end link every 40s. Therefore, we instead consider a *generate-when-requested* network. In such networks, we do not assume that end-to-end entangled links can be stored between any two scheduled periods of time. This type of network is implementable with the technological maturity of current devices and those that will exist in the coming years.

B. RELATED WORK

Our architecture is designed to be compatible with the network application execution environment Qoala, which is described in [16]. Qoala is an extension to the QNodeOS operating system for quantum network end nodes developed by Delle Donne *et al.* in [15]. When using the Qoala execution environment on QNodeOS, it is assumed that there exists a network schedule in order for entanglement generation to occur. One of the aims of this work is to design an architecture which can produce suitable network schedules which enable the execution of quantum network applications using Qoala on QNodeOS.

Our work is also compatible with the network stack proposed by Dahlberg *et al.* in [24]. In particular, the network schedules that our architecture produces replace the queue used in the implementation of [24] in [23].

We build on the formalism presented in [25]. In particular we incorporate the framing of entanglement generation as a problem in the field of scan statistics (see e.g. [26], [27]) into our architecture.

There are several other authors who have proposed architectures for quantum networks, however they each have a different scope for the architecture than ourselves. In particular, none of them explicitly consider the influence of requirements of executing local applications on the network.

For instance, Skrzypczyk *et al.* in [28] propose an architecture around using TDMA schedules to generate good quality entanglement. Whilst we build on their ideas about scheduling, they do not consider how their scheduling will impact the ability of end nodes in the network to execute the applications requiring this entanglement, nor how the demands are generated from the applications.

Cicconetti *et al.* in [29] and Gu *et al.* in [19] consider the problem of scheduling requests for entanglement generation in a quantum network. However in both cases they consider a pre-loaded network rather than a generate-when-requested network. Furthermore, they do not consider how the end nodes use the entanglement which is generated, and also consider a system where every edge in the network is in constant communication with the central controller, whereas we only require sporadic communications. A disadvantage of this compared to our approach is that due to latency when communicating between the edges and the central controller, there is an inherent loss in the quality of entangled links which can be created. Furthermore, they only consider requests for single entangled links, whereas we consider requests for packets of entangled links.

Van Meter *et al.* in [30] propose an architecture which focuses on routing of entanglement across many smaller networks, and the protocols required to do so. This again does not account for the local nodes, nor does it consider the interaction of scheduling entanglement generation on the network with executing the applications on end nodes. Furthermore, the scope of our work is to provide an architecture for a single quantum network which can be centrally controlled, rather than for a quantum internetwork.

Our architecture also has similarities to a software-defined network [31]–[33]. In particular, we see our work as a method of facilitating the implementation of a quantum SDN. There have been several examples of prior work on defining a quantum SDN. For example in [34], Kozlowski, Kuipers and Wehner give an implementation of a quantum SDN using the P4 language. However, this implementation only focuses on the network aspects, and does not consider the execution of applications on the end nodes. There have also been quantum SDNs proposed and demonstrated by Yang and Cui in [35] and proposed by Chung *et al.* in [36]. These works do not explicitly consider scheduling at the end nodes in the network, and they focus on demands being registered via a web-interface. In contrast we create a fully-autonomous architecture, where the end nodes, rather than the users themselves, submit the demands in response to users wishing to run specific applications.

There have also been several SDN architectures proposed for quantum key distribution (QKD), e.g. [37]–[46]. However, our architecture can support arbitrary applications, rather than just QKD. This imposes more constraints on the process of generating entanglement than are considered in these works. For the same reason we cannot use the demand format in [47]. Furthermore, we are focused on the execution of these applications rather than adding extra security to a classical network as in [37].

Our architecture also has similarities to a so-called *Time-triggered Ethernet (TTEthernet)* architecture [48]. However, for our problem one cannot just use a classical TTEthernet system for a couple of reasons. Firstly, as a classical system the literature does not directly take into account quantum-specific constraints such as decoherence. Secondly, the predominant usages of TTEthernet in the classical sphere are in control systems, for example in spacecraft [49], [50], where the sources of (time triggered) demands on the network are known *a priori* and can therefore be accounted for. However, in our model the network does not know where demands will come from, what resources or objectives they will require, or for how long they will need to use the network.

III. DESIGN CONSIDERATIONS

Our network architecture defines a framework for integration of a network schedule with the execution of quantum programs running on end nodes. The design of the architecture thus inherits considerations pertaining to robust operation of a quantum network (see e.g. [24]) and considerations from the application execution environment of end nodes [16]. We describe at a high level each of the relevant principles and highlight how they may be consistently combined into the foundational pillars of a single network architecture.

A. NETWORK

1) Devices and Components

We consider a quantum network comprised of four types of devices. Figure 1 illustrates an example of how these components can fit together. The first device type, called *end nodes*, execute quantum applications, operate under independent (local) control, and accept input from users. An end node may be a processing node with some memory capabilities, such as in [51]–[58], or a device which is capable of preparing/measuring single photons, such as in [59], [60]. Any end node can also perform classical operations, such as arithmetic operations and classical communication.

End nodes may be connected to a second type of device, *metropolitan hubs*. These are devices which enable pairs of nodes located close together (typically <50km end-to-end) to create entangled links. Examples of such devices are entanglement distribution switches [61]–[63] and quantum routers [64], which employ quantum memories at the hub, or alternatively entanglement generation switches [65], [66], which do not rely on quantum memories at the hub. As quantum routers are physical devices only, a metropolitan hub based on a quantum router must be paired with a simple operating system for coordinating entanglement swapping. A metropolitan hub will typically have many devices connected to it, including both end nodes and junction nodes.

The third type of component we include are *repeater chains*, which allow for long distance entangled links to be created between two parties. Repeater chains are made from a linear chain of *repeater nodes*, such as those in [13], [67]–[70]. We treat repeater chains as a single network component

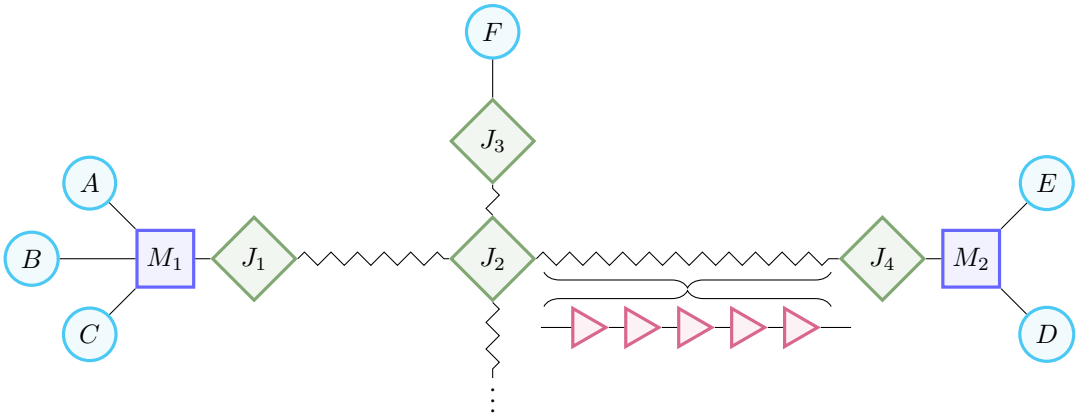


FIGURE 1: A general quantum network may be built from four kinds of devices: end nodes, metropolitan hubs, repeater chains, and junction nodes (see III-A1). These devices may be connected in the manner illustrated. Circles A-G represent user controlled end nodes, squares M_i are metropolitan hubs and diamonds J_i are junction nodes. Repeater chains are represented by zig-zag lines, while individual quantum repeater nodes are represented by triangles.

which can be configured to produce links between two border nodes connected to either end of the repeater chain at a fixed rate and average fidelity.

The final component we consider in the network are *junction nodes*. These provide an interface between multiple repeater chains and between repeater chains and a metropolitan hub. As an interface, junction nodes may be instrumental in essential processes for connecting heterogeneous quantum devices, such as waveform matching and entanglement buffering. Such nodes remove the requirement to have direct repeater chains between every pair of metropolitan hubs. Junction nodes may be implemented using a combination of one or more of the previous devices. We refer to repeater chains, metropolitan hubs and junction nodes collectively as *internal* components, and end nodes as *external* components.

We assume that each internal component of the network has a control API which can be used to install network schedules. Furthermore, such an API is able to expose information about the operational parameters of the component. For example, a metropolitan hub may expose the maximum number of node pairs it can connect simultaneously; a repeater chain may expose the rate and fidelity at which entangled links between the end points are created; and junction nodes may expose the number of links which can be buffered and for how long.

2) Hardware Agnosticism

Each of the multiple possible implementations of any network device is associated with specific requirements relating to its operation. However, a network architecture should be able to seamlessly support whatever hardware is being used. As a result, where necessary we assume that the control API of internal network components and the application execution environment of end nodes make concessions for the specific requirements of a device, ensuring correct functioning. Thus our network architecture is hardware agnostic and compatible with a heterogeneous network consisting of devices based on a variety of different hardware types.

3) SDN Controller

Software defined networks (SDN)s [31]–[33] separate the data plane of a network, which forwards traffic to the appropriate destinations, from the control plane, which makes decisions about how traffic should be handled. Decisions taken by the control plane include routing and resource access management. The SDN framework aims to simplify network management and make networks flexible and cost-effective. We consider the traffic on the data plane of a quantum network to consist of point to point attempts to generate entanglement.

We assume that the quantum network follows the general architecture of an SDN. In particular, we assume that the network is overseen by a (logically) central controller, similar to for example [11], [28], [34], [38]. Such a controller has the authority to compute and enforce *network schedules*, which dictate when entangled links can be generated for particular pairs of end nodes.

We also assume that the central controller has a complete overview of the entanglement generation capabilities of the components of the network (consideration III-A7). However, we do not assume it knows whether or not any given attempt to generate an entangled link succeeds or fails.

4) Device Autonomy

We assume that each component of the network is able to operate without direct interaction with the central controller. In particular, we assume for each component in the network there is a local controller which handles executing an installed network schedule on that component without further input from the central controller. On end nodes, this local controller takes the form of an execution environment

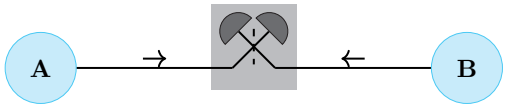


FIGURE 2: Example of an entanglement generation protocol (heralded entanglement). Nodes A and B probabilistically send photons to a 'heralding station' located midway between them. This heralding station consists of (single) photon detectors and a beamsplitter. Depending on the pattern of photons detected, it is possible to determine if an elementary entangled link has been generated, and to subsequently inform the nodes. More details can be found in, for example, [13].

such as Qoala [16] running on QNodeOS [15]. On internal components, this local controller may be more limited and simply implement a rule-set which realises the network schedule. For example, if a repeater chain is required by the network schedule to produce entangled links with certain rate/fidelity characteristics, then we assume there exists a (logically) centralised local controller which implements some policy, e.g. [12], [71], on the repeater chain in order to create the required entangled links.

5) Protocols for generating entanglement

Our architecture is agnostic to how entangled links are produced at the physical layer. However, we assume that any protocol which is used allows the rate and fidelity at which entangled links can be produced to be calculated and exposed. We also assume that entangled links between neighbouring nodes are created using a heralded entanglement generation scheme such as [13], [67] (Figure 2). In particular this means that the nodes attempting to create an entangled link receive a success or failure outcome, indicating when an entangled link has been created. Hence subsequent attempts may be triggered conditioned on the success or failure of previous attempts.

We refer to entangled links between end-nodes as *end-to-end entangled links*, and entangled links between neighbouring nodes as *elementary entangled links*.

Note that in the literature an *entanglement generation attempt* typically refers to an attempt to generate an elementary entangled link, following some specified entanglement generation protocol, for example as in Figure 2. However, when referring to entanglement generation attempts we will mean attempts to generate end-to-end entangled links, requiring the use of all internal components along a specified path and employing a pre-determined protocol. For example, attempts to generate entanglement between nodes A and D in Figure 1 require the simultaneous use of resources at M_1, J_1, J_2, J_4 and M_2, as well as the repeater chains connecting these components.

6) Network Stack

A network stack is a layered set of protocols and services that work together to enable network communication. Each layer in the stack is responsible for a specific aspect of network functionality. Together the layers provide a comprehensive framework for data transmission across classical networks [72] or delivery of end-to-end entangled links by quantum networks [24].

We assume that the underlying network stack is as proposed in [24]. In particular, we assume that the link layer is implemented using the protocol in [23]. This protocol assumes the existence of a schedule computed by an external scheduler which determines when entanglement generation can take place.

Our architecture facilitates the construction of such schedules, following a synchronous time-slot scheduling approach with variable length time-slots. We define a periodic timeline for computation and distribution of network schedules to network components. A network schedule is a time ordered plan for the execution of finite duration network tasks. We provide a precise system for translating the demands for service originated by applications running on end nodes into the task executions that make up a schedule. The translation process takes into account the capabilities of the various network components, learned through a capability update process between components and the central controller.

7) Network Capabilities

The network provides information about its capabilities to the end nodes as a list of pairs $\texttt{(rate, fidelity)}_{i,j}$, describing the rate and fidelity respectively at which entangled links can be generated between a pair of end nodes i and j. These pairs can also be endowed with extra information about, for example, the expected jitter and the availability of each option. We assume that the SDN controller for the network is able to compute these properties from its knowledge about the capabilities of each individual component and the state of the network.

8) Timing

The time required for network elements to complete actions can only be estimated with finite precision. At the physical layer, actions have precisely characterised durations, allowing for accurate synchronisation between multiple nodes, with timing precision ranging from tens of picoseconds (ps) to microseconds (μs), depending on the operations [52], [73]. Precise timing of a sequence of operations is crucial for processes like entanglement generation [53], [54], [57], [58], [74]–[76].

In contrast, at higher layers of the network stack, actions have variable durations and latencies, limiting feasible timing precision to μs or milliseconds (ms). For example, transmitting a network schedule over the internet from a central controller to a node 10 km away may take from 50 μs to several ms. This variability in process duration can be due a variety of sources. Typical examples include that a single

process may comprise a large number of computational (low level) operations, that there may be traffic from other processes–creating competition and leading to waiting times for limited computational resources or network bandwidth, or inter-process interactions that necessitate waiting for responses from inter-dependent processes [77]–[79]. To accommodate this variability, our network architecture uses a modular framework, computing and distributing schedules to nodes in advance.

An additional type of timing consideration is that entanglement generation involves sequential non-overlapping attempts, each with a particular sequence of operations, setting a minimum period and maximum rate for the process. This also means operations cannot change mid-attempt without disrupting entanglement generation.

B. APPLICATIONS

For simplicity, we assume quantum network applications are executed between two end nodes. However, our architecture is directly compatible with applications involving multiple end nodes.

1) Execution Environment

To execute applications, end nodes require a runtime execution environment. In this work we assume the Qoala [16] runtime environment is employed, running on QNodeOS [15]. Qoala breaks applications down into a *program* running on each end node. Each of these programs is then broken down further into *blocks* of instructions, which can be one of four types: classical local (CL), classical communication (CC), quantum local (QL) or quantum communication (QC). Quantum communication blocks correspond to generating entangled links.

When an application is executed, each block of instructions causes a task to be created, which is then scheduled for execution by a local scheduler on that end node. The execution of the task corresponds to realising the block of instructions. Task execution scheduling can either be performed in advance or at runtime.

The Qoala environment comes equipped with a compiler which runs locally on each node. This compiler provides advice about how local hardware parameters (e.g. memory lifetimes) are mapped to requirements on any entangled links produced. The compiler also produces metadata for every block and for an entire program. This metadata covers any constraints on the execution of tasks corresponding to the constituent blocks of a program. This metadata is used in a process called *capability negotiation* during which the nodes ensure that the various programs comprising the application will be executed in a compatible manner. Whilst here we assume use of the Qoala environment, our architecture is compatible with any runtime environment with an equivalent notion of task scheduling and a compiler that provides equivalent metadata.

2) Application Classes

There are many different quantum network applications, and each will have different requirements for entangled links generated by the network. However, they may be broadly grouped into two different classes, *measure-directly* and *create-and-keep* [24]. In measure-directly (MD) applications, qubits are measured as soon as entanglement is produced, and no states are kept in memory. In this case, the demand for the generation of entangled links is elastic [80]. Examples of MD applications include *quantum key distribution* (QKD), which facilitates provably secure secret sharing [1], [2], and *deterministic teleportation* [81], [82].

In create-and-keep (CK) applications, qubits are stored in memory after entangled links have been generated. Typically CK applications will require many qubits to be stored in memory simultaneously. This places limitations on the spacing between the generation of new links. An example of a CK application is *blind quantum computing* (BQC), which facilitates secure remote computation (e.g. [3]–[5]).

3) Architecture

Our architecture is agnostic to the architecture of the end nodes of the network. However, for convenience we assume that processing nodes have a single monolithic *quantum processing unit* (QPU), which is used both for entanglement generation and performing local gates and measurements. Such a QPU is assumed to be capable of performing only a single operation at any given time. This is in line with state-of-the-art implementations of end nodes [23], [53]. We do not make any assumptions about the classical processing capabilities of the end nodes.

4) Knowledge of the network

We assume that an end node has minimal knowledge about the rest of the network. In particular, we assume that an end node only knows the capabilities and status of their own hardware, the identities of any neighbouring network components and the identity of any control devices in the network. Furthermore, we assume that an end node only knows the programs which are running on itself. In particular, it is not required that an end node knows the precise program of the other node in the application. Additionally, each program on a given end node should be independent, and the ability to execute a program should not require knowledge of the other programs running on the end node.

IV. NETWORK ARCHITECTURE

Our proposed network architecture comprises on-demand processes triggered by end nodes and periodic processes hosted by the central controller. The on-demand processes are a *network capability update*, *capability negotiation*, and *demand submission*. In the network capability update and capability negotiation the end nodes learn information about each other and the network in order to construct a unified demand. Demand submission is the process by which end nodes submit their demands to the central controller. The

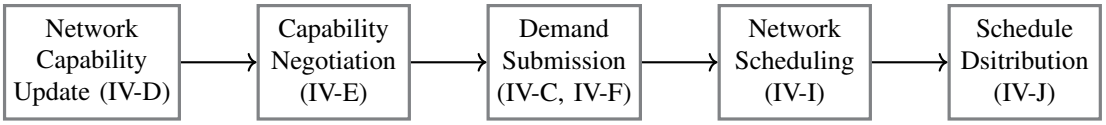


FIGURE 3: Flow of information in the architecture. The processes of 'Network Capability Update' and 'Capability Negotiation' allow the nodes to gather enough information to be able to submit a unified demand. These demands are then used to construct a central network schedule, which in turn is used when constructing the local node schedules.

periodic processes are *network scheduling* and *schedule distribution*, where the central controller computes the network schedule before distributing it to the end nodes respectively. The high level flow of information through these processes is summarised in Figure 3. Example timings of the distinct processes are illustrated in Figure 4. A detailed overview of our architecture is illustrated by the interaction diagram in Figure 5.

Our architecture introduces several unique features which ensure compatibility of the architecture with end nodes and program execution. Firstly, we periodically distribute network schedules to end nodes well before the start time of the network schedule, which allows nodes enough time to compute local schedules that incorporate the network schedule. Secondly, we introduce a demand format which takes into account local processing time requirements. This demand format includes a precise description of the entangled links which are required by an application, through a notion of a *packet of entanglement*. Finally, the format and construction of our network schedules ensure that end nodes are certain about when they can attempt to generate entanglement, allowing for efficient and effective program execution.

In the rest of this section, we first introduce several preliminaries which precisely define how an application's requirements translate to a demand for service from end nodes to the network. Then, we provide a detailed description of each stage of the network architecture.

A. PERIODIC COMPUTATION AND DISTRIBUTION OF SCHEDULES

The essential task of computing and distributing a network schedule is executed periodically by the central controller. Each schedule is associated with a version identifier and covers an identical execution time known as the *scheduling interval* (denoted T_{SI}). Periodic triggers for computation and distribution of the schedule are defined with respect to the T_{SI}. Figure 4 illustrates possible trigger timings for the components of this periodic task and illustrates the back-to-back execution of subsequent schedules.

The primary advantage of a periodic approach to scheduling, as compared to various possible on-line approaches to scheduling, is that it limits the number of updates to the schedule that may be triggered. In particular, the central controller does not need to update the schedule every time a new demand is submitted, which reduces the number of time-consuming interruptions during scheduling. This approach is also advantageous for end nodes, which receive only finalised network schedules. This approach avoids unnecessary interruptions to local schedules and local program execution. Furthermore, any buffer between the time at which a schedule is received and the time it takes effect allows end nodes to optimise their local program schedules without compromising execution of the network schedule.

Compared to on-line network scheduling with an unbounded number of updates, the periodic distribution of pre-computed network schedules reduces the number of messages which need to be sent by the central controller. As the loss of a message from the central controller can lead to a disruption of service, this helps to improve the reliability of the network. With this approach to scheduling, all the components of the network know a) when the schedule is changing and b) which schedule version should be in effect when, reducing the amount of communication required between the end nodes and the central controller. These features reduce the likelihood that network components attempt to execute different versions of the network schedule.

B. APPLICATION SESSIONS

As the outcome of many quantum applications is probabilistic, a single application needs to be executed many times to extract a useful and reliable output. We refer to each of these individual executions of an application as an *application instance*. Nodes will often require that all these instances are executed before some time elapses. For example, suppose Alice and Bob wish to communicate securely within the next two hours. To do so, they may use QKD to generate a raw key and extract a secret key which can then be used to communicate. To generate enough raw key would then require them to execute many instances of the QKD application during the next two hours. To capture these requirements, we define an *application session*.

Definition IV.1 (Application Session)**.** Suppose end nodes $\mathcal{N} = (\texttt{node1}, \texttt{node2}, \ldots)$ wish to execute application $\texttt{App}$ at least $N_{\texttt{inst}}$ times, before time $t_{\texttt{expiry}}$.

Then we write the corresponding *application session* as

$$\mathcal{S} = (\mathcal{N}, \texttt{App}, N_{\texttt{inst}}, t_{\texttt{expiry}}) \tag{1}$$

Application instances are not required to be identically executed. For example, in an application such as verifiable blind quantum computing, (e.g. [83]), some instances may

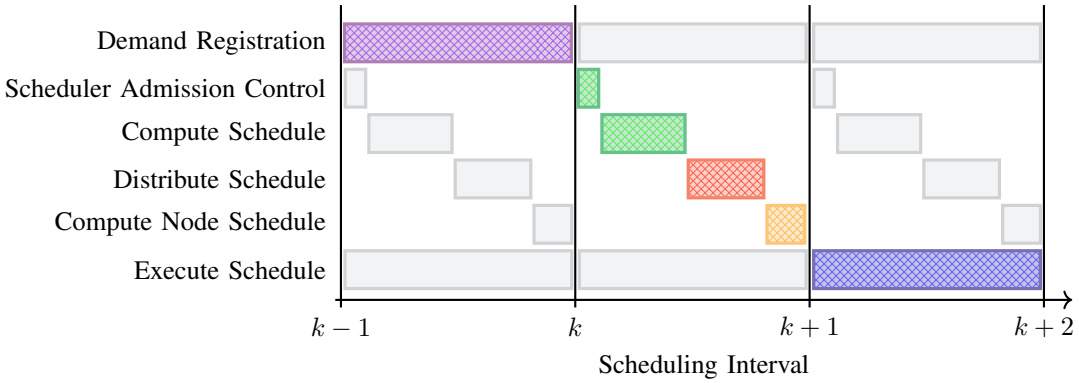


FIGURE 4: Example of the timings for computing, distributing and executing the kth network schedule (in colour and hatched). In grey and unhatched are the corresponding operations for preceding and subsequent network schedules. The process of registering received demands is continuous, however we indicate here the time during which if a demand is submitted, then it will be first considered for scheduler admission as part of computing the kth network schedule.

correspond to computation rounds (where the calculation is performed) and others to test rounds (where the veracity of the server is checked). However, all instances should use the same programs on each end node.

We expect that the specified expiry time, $t_{\texttt{expiry}}$, will be much longer than any time-scale of the network. For instance, if the network updates (see §IV-I) on the order of minutes, we expect $t_{\texttt{expiry}}$ to be on the order of hours.

When pairs of nodes send their demands for entanglement to be generated to the network, they will do so on a per-session basis, rather than for each individual instance. This reduces the number of demands that the central controller has to process. Furthermore, it gives the central controller an overview of the scope of a demand, and therefore it can make better decisions regarding whether and how to give it service.

1) Executing Application Instances and Obtaining Minimal Service

It will be useful to be able to talk about whether program instances have executed successfully. Due to the imperfect nature of qubits, even if all gates are performed perfectly, an 'incorrect' outcome may be obtained, which could be interpreted as an unsuccessful execution. To clarify what is meant when we talk about application instances successfully executing, we make the following series of definitions:

Definition IV.2 (Application Instance Execution)**.** We say an instance of an application is *executed* when at least one of the blocks of instructions are performed by the relevant processor.

Definition IV.3 (Successful Quantum Block Execution)**.** We say a block consisting of local gates *executed successfully* if all included gates were applied to the qubits without destroying them and a signal confirming this is passed back to the local scheduler. We say a quantum communication (entanglement generation) block *executed successfully* if a *packet of entanglement* (see §IV-C1) is generated, and a signal confirming this is passed to the local scheduler.

Definition IV.4 (Successful Application Instance Execution)**.** We say an application instance *executed successfully* if all blocks are successfully executed, including receiving confirmation signals.

Definition IV.5 (Quantum Successful Execution)**.** We say an application instance achieved *quantum success* if it was successfully executed and the correct outcome was obtained.

Definition IV.6 (Minimal Service)**.** We say a session obtains *minimal service* if $N_{\texttt{inst}}$ instances of the application are successfully executed before the session expiry time $t_{\texttt{expiry}}$ elapses.

Note that in the literature, what we refer to as 'quantum successful execution' is commonly referred to as "application success", for example, as in [16]. Our fine grained terminology makes it possible to address the outcomes of individual application instances and distinguish between an instance successfully executing and an instance achieving quantum success.

The aim of the network is to ensure that as many sessions as possible obtain minimal service. To do so, it facilitates the successful execution of application instances by scheduling time to create packets of entanglement. In contrast, the aim of end nodes is to achieve quantum successes from successfully executed instances by suitably scheduling the execution of local gates and operations.

We also define a notion of 'load' on the network, corresponding to the number of pairs of users which have submitted demands for entanglement generation. In particular, if the load on the network is high, this should correspond to a decrease in the likelihood of a session obtaining minimal service from the network, due to the network resources not being able to serve all submitted demands simultaneously.

C. DEMAND FORMAT

The network should be agnostic to the particular applications being executed on end nodes, and only make decisions based on their requirements in terms of entangled links. Therefore, application sessions should create *demands* which can be submitted to the network. These demands allow the nodes to communicate their capturing requirements without revealing the particular application being executed.

To facilitate the central controller creating a schedule for the generation of entangled links, the the demand should communicate two sets of information to the central controller. Firstly, the central controller needs to know the quantity and quality of entangled links that must be produced. Secondly, the central controller requires knowledge of the frequency with which and over what time period these entangled links must be produced.

1) Packets of Entanglement

We first address specification of the quality and quantity of entanglement required. Recall from §III-B2 that when executing an application from the create-and-keep class, each instance will require many co-existing entangled links. Each of these links cannot be too old because quantum memories, which store them, exhibit time-dependent decoherence, causing the quality of the links to decrease over time. The maximum age of a link depends both on the hardware of the specific end nodes and the initial fidelity of the entangled link. One approach to satisfy these requirements is to strictly control the jitter between generated entangled links, for example as in [28]. However, we instead consider the generation of *packets of entanglement*, rather than individual links. The generation of a packet of entanglement corresponds to the co-existence of all required entangled links for an instance of an application. As each instance of an application is independent, there is no time constraint between the generation of each packet of entanglement. Therefore, following the definitions in [80], any demands, both from sessions of measure-directly and of create-and-keep applications can be treated as elastic.

One may also attempt to control the jitter between the generation of packets, though the ability to do so will depend on the network scheduling algorithm employed. Although jitter may have an impact on node schedules, jitter control is not necessary to enable execution of application instances.

The following definitions, motivated by the formalism set out in [25], precisely specify what is meant by packets of entanglement.

Definition IV.7 (Window)**.** A *window* in the context of entanglement generation is the longest time that an entangled link can be kept in memory without decohering too much to be useful. If a link has been in memory for longer than the length of the window it is discarded. This means that all the entangled links required to execute an instance of a quantum application need to be generated within a window's duration of each other.

Definition IV.8 (Entanglement Packet)**.** A *packet of entanglement* or *entanglement packet* for a given application session is the tuple $(w, s, F_{\min})$, where w is the window within which all entangled links must be created, s is the required number of entangled links and $F_{\min}$ is the minimum fidelity new links may be created with.

To see how this is useful, suppose Alice and Bob wish to execute two instances of an application, each requiring three entangled links. Further suppose that Alice and Bob's hardware is such that each entangled link can only be stored for 0.5s. Instead of saying that Alice and Bob require 6 entangled links with expected fidelity F, it is much more precise to say they need two $(0.5s, 3, F)$ packets of entanglement. In particular, if the network allocated resources such that Alice and Bob sequentially generated 6 entangled links, each 1 second apart from the other, this would satisfy the former requirement, but Alice and Bob would still not be able to achieve quantum success when executing their applications. By specifying the packet of entanglement required, the network can allocate resources for sufficiently long periods in the network schedule that the required entanglement, i.e. a packet, can be produced. The packet formalism thereby ensures that application instances can actually be executed.

Definition IV.9 (Packet Suitability)**.** A packet of entanglement is *suitable* for an application if the existence of entangled links adhering to the form of the packet would allow an instance of the application to be executed with an acceptable probability of achieving quantum success.

It is possible for an application instance to have multiple suitable packets of entanglement. This may arise from, for example, accepting links which are generated with a lower expected fidelity and shortening the window to compensate. The end nodes could also include extra links which may not be directly required by the protocol. End nodes include a finite subset of all possible suitable packets in their demand. We write $\mathcal{P}$ for the set of suitable packets included in the demand.

In order to compute the set of suitable packets, the nodes need to know what quality of entanglement the network can produce, as well as the hardware on the other node(s). These pieces of information are obtained in the *network capability update* and *capability negotiation* processes respectively. Given this, they can establish how long the links can be stored in memory, and thus construct the set of suitable packets.

2) Timing Constraints

We now address communicating the frequency and time-period over which packets of entanglement are produced. For each suitable packet of entanglement, the nodes specify an average rate R at which they wish such packets to be produced. This rate can be set to 0, which indicates to the central controller that the nodes will be satisfied with the

minimum possible rate of packet generation which almost certainly ensures the session obtains minimal service. In this case, the nodes also submit N_{inst} as part of the demand, so the network can calculate this minimum rate when the demand is accepted for scheduling.

The central controller will define a *service model* which specifies how the requested rate will be treated. For example, the rate may be met exactly, or alternatively the network may increase or decrease the rate at which packets are generated within specified limits, depending on the current network load.

Alongside the packets and associated rates, the nodes also submit two further pieces of timing information. Firstly, they include the expiry time of the session $t_{\texttt{expiry}}$. Secondly, they include a minimum separation between attempts to generate packets of entanglement. This minimum separation is included to ensure that there is sufficient time for local operations to be performed before the next allocated period of time for generating entangled links begins. Local operations may either be additional blocks of quantum operations in the application program or operations to reset the hardware between subsequent attempts to generate a packet.

Submitted rates may depend on factors such as how often the nodes intend to execute instances of an application, as well as pre-existing device agreements with the network. Thus, the values are determined as part of the *capability negotiation* phase. Determination of the minimum separation required between attempts to generate a packet and an appropriate expiry time also requires input from both nodes, and as a consequence these values are set during *capability negotiation*.

3) Full Demand

Combining all of the above we obtain the demand which end nodes submit to the network:

$$\mathcal{D} = \left(\left\{ (w, s, F, R)_p \right\}_{p \in \mathcal{P}} ; t_{\texttt{minsep}}, t_{\texttt{expiry}}; N_{\texttt{inst}} \right), \quad (2)$$

where $\mathcal{P}$ is the set of submitted suitable packets.

As the demand format needs to be compatible with all end nodes, the time dependent parameters $w, R_{\texttt{packet}}, t_{\texttt{minsep}}$ and $t_{\texttt{expiry}}$ should be specified in terms of real-time units, such as seconds or per-second as applicable, rather than in terms of local or network time slots. The central controller can then convert them to a notion of time-slots, if required, when computing and distributing the schedule.

D. NETWORK CAPABILITIES UPDATE

To enable construction of the suitable packets of entanglement, the end nodes need to know whether the network can generate entangled links between them, and with what quality these links can be produced. Providing the nodes with this information is the primary objective of the *network capabilities update* process of our architecture (interaction **A** in Figure 5).

The application stack obtains this information in the following manner. Firstly, a query is sent to the quantum network agent (QNA) (see Figure 5). This query can be either for an overview of the network, or for the specific entanglement generation capabilities with another party. If the QNA has recently obtained the requested information, then it responds directly, otherwise the query is forwarded to the *network capabilities manager* of the central controller.

In the case of an overview request, the response includes with whom entanglement can be generated, as well as general information about the status of the network such as the current load. If the application stack requests the capabilities for a specific node, this information is returned in the form of (R, F) rate-fidelity pairs describing the rate at which end-to-end links can be generated with fidelity F.

In order to determine the quality of entanglement which can be generated, the central controller performs the following tasks: First, it establishes along which paths through the network entangled links can be generated. Then, along these paths the central controller devises a scheme which will enable end-to-end entangled links to be produced. Using the information which the central controller has about the fidelity of the entangled links which can be produced along each of the segments of this path, the overall end-to-end fidelity can be determined and communicated back to the requesting end node's quantum network agent.

The central controller retains the right to be selective about which possibilities it communicates. For instance, it may discount certain paths or configurations which would put excessive pressure on the network. Furthermore, the network only communicates the quality of entanglement and *not* the paths back to the nodes. This is to give the controller flexibility, where possible, to create the same quality links using multiple different paths as well as to maintain the agnosticism of the end nodes as to the internals of the network.

E. CAPABILITY NEGOTIATION

Before any application can be run, or a demand submitted to the network, the nodes must align amongst themselves exactly how the application will be run. Ultimately the aim of this is to create an application session and corresponding demand, as well as to finalise any metadata about the programs which still needs to be set. To do this, the nodes carry out *capability negotiation* (interaction **B** in Figure 5).

During capability negotiation end nodes exchange relevant information, for example the quantity and quality of qubits which can be made available and how other end nodes need to interact with their programs. In combination with the data from a network capability update, the end nodes should be able to exchange sufficient information to determine the acceptable packets and calculate/decide upon the packet generation rates they will request from the network. By the end of this exchange, the end nodes will have set values of $N_{\texttt{inst}}, t_{\texttt{expiry}}$ and $t_{\texttt{minsep}}$. Once capability negotiation has concluded, up to classical communication in the application

itself, the end nodes should be able to execute their programs independently and without further classical communication.

F. DEMAND REGISTRATION

When demands are received by the central controller (interaction **C** in Figure 5), they undergo an initial registration process. If a demand passes this process, then it is placed into the *demand queue* for consideration by the scheduler admission control. Otherwise, it is immediately rejected by the network. The end node which submitted the demand is informed of the outcome of the demand registration process, and additionally can be informed as to the reasons for rejection. In the case no such acknowledgement arrives, end nodes should assume that the demand has been rejected.

The main aim of this process is to filter out any obviously infeasible or unreasonable demands. On top of this, this process may also reject demands based on the load which the network is experiencing. For example, if load on the network is high, then the demand registration process may also immediately reject demands with a high expected queuing time. The precise rules which the demand registration process implements will depend on the network implementation and the types of behaviour which the network operator desires.

1) Leaving the Demand Queue

The central controller determines how and when each demand leaves the demand queue. The positive outcome for a demand is that it passes the scheduler admission control, and is accepted to be scheduled. Depending on the load on the network and the nature of the demand, this may happen at the start of the next scheduling interval. Alternatively, a demand may possibly be held in the queue for several scheduling intervals, until there is sufficient capacity for the network to serve it.

Alternatively, demands may be removed from the demand queue. If a queued demand reaches it's expiry time, it will be marked as expired by the central controller and removed from the queue. Demands may also be removed from the queue before they expire if the central controller evaluates that removing the demand is beneficial to the overall performance of the network. For example, the central controller may apply a rule where any demand which would have failed demand registration had it been submitted at the current time is removed from the demand queue. As with demand registration, the exact rules governing premature removal from the demand queue should be specified in an implementation.

G. SESSION INITIALISATION

Following capability negotiation and demand submission, each end node performs some initial configuration (interaction **D** in Figure 5). In particular, the blocks of instructions that make up the program are submitted to the local scheduler, and any initial configuration of the end node's quantum network stack is performed. An example of configuration of an end node's quantum network stack which may be required is the establishment of a quantum network socket for the application, as in [16], [73], ensuring that generated entanglement is assigned to the correct application.

H. PROCESSING DEMANDS AND PACKET GENERATION TASKS

While the demand format is sufficient for communicating the requirements of the nodes to the central controller, it is not an efficient format for use by a network scheduling algorithm. To address this, when a demand is considered by the scheduler admission control it is converted into a *packet generation task* (PGT). This is an internal representation of the demand containing only the information required to construct the network schedule.

1) Finite Execution Times

Due to the probabilistic nature of generating entangled links, it is impossible to guarantee that a packet will be generated when any finite execution time is allocated to a *packet generation attempt* (PGA). Therefore, the network only guarantees that a packet will be generated in a given PGA with some probability $p_{\texttt{packet}}$. This probability is an internal parameter, known only to the network. The rate at which PGAs are scheduled may be increased to compensate for $p_{\texttt{packet}}$. The value of $p_{\texttt{packet}}$ can either be static, or can alternatively be determined using a method conforming to Algorithm 1. Once the value of $p_{\texttt{packet}}$ has been determined, the execution time of each PGA, E can be determined, using a method conforming to Algorithm 2.

In the process of selecting $p_{\texttt{packet}}$, there is a trade-off between the length of each PGA, E, and the rate at which PGAs need to be scheduled, $R_{\texttt{attempt}}$. As the value of $p_{\texttt{packet}}$ increases, the required length of the PGA increases, whilst the required rate of PGAs decreases. Furthermore, as E and $R_{\texttt{attempt}}$ change at different rates with $p_{\texttt{packet}}$, the resource utilisation $U_\tau := E_\tau R_\tau^{\texttt{attempt}}$ for a given packet generation task τ is not constant with $p_{\texttt{packet}}$. We will see in §IV-I1 that the resource utilisation of packet generation tasks is an important quantity that determines how many or which demands can be accepted by the scheduler admission control.

Algorithm 1: Determining $p_{\texttt{packet}}$

Input : Network state, service agreements, w, s
Output: Probability of generating a packet in a PGA, $p_{\texttt{packet}}$

1 Determine the value of $p_{\texttt{packet}}$ for given w, s given network conditions and adhering to service agreements.

2) Setting the rate of packet generation attempts

The central controller must determine a suitable rate, $R_{\texttt{attempt}}$, at which to schedule PGAs. If the nodes request

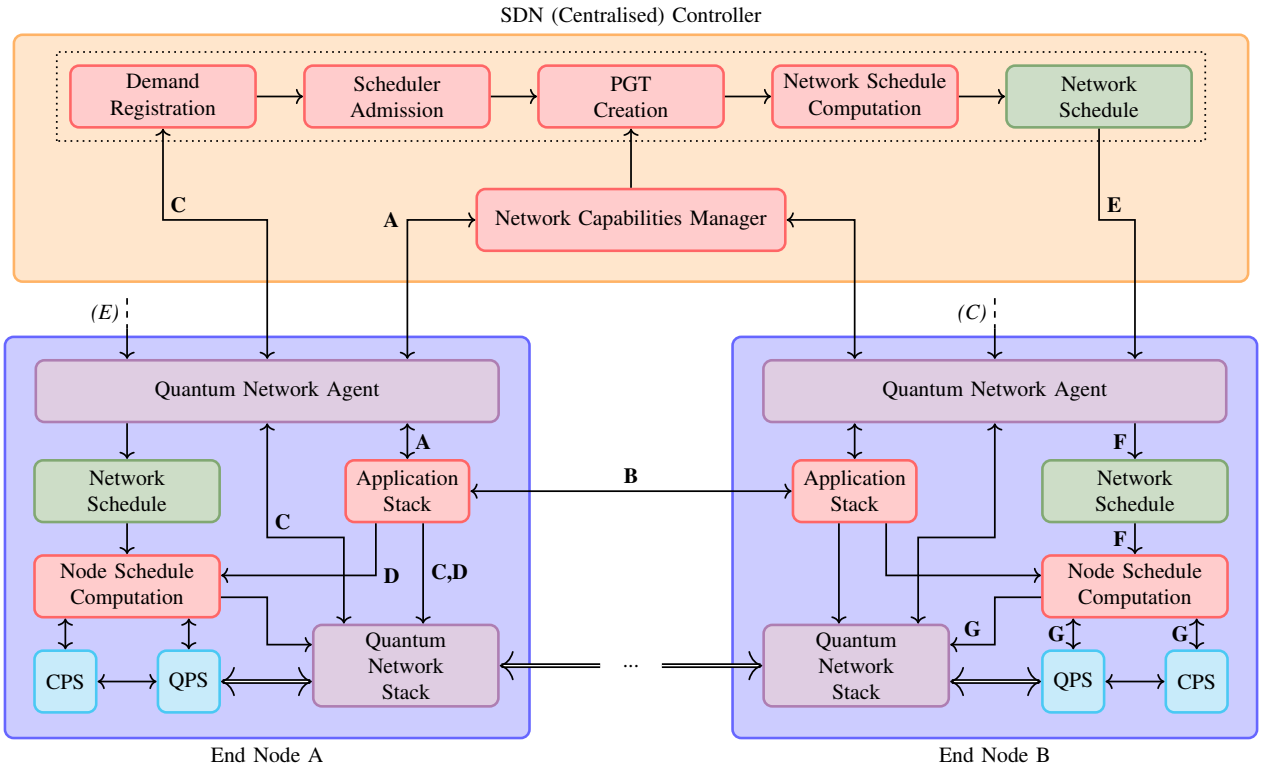


FIGURE 5: Interaction Diagram for our proposed quantum network architecture. Elements in red are local software components. Single stroke arrows represent purely classical interactions and double-stroke arrows represent quantum interactions. The dotted arrows denote the corresponding interaction from the other end node. The *QPS* and *CPS* are the *Quantum Processing System* and *Classical Processing System* respectively. The application stack includes the application code, compiler and execution environment. The *Network Capabilities Manager* is an oracle which can be queried by end nodes to find out information about the network as part of the network capabilities update phase of the architecture (IV-D). The process of *Scheduler Admission* includes the demand queue. The *Quantum Network Stack* is that of [24]. The ellipsis represents the quantum network. The labelled interactions are as follows: **A**: Network Capability Update; **B**: Capability Negotiation; **C**: Demand Registration; **D**: Session Initialisation; **E**: Network Schedule Distribution (note this goes to all components of the network); **F**: Input of network schedule into the local schedule; **G**: Execution of the schedule(s).

Algorithm 2: Determining the length of PGAs

Input : w, s, $p_{\texttt{succ}}$, $p_{\texttt{packet}}$
Output: Length of a PGA, E

1 Calculate the shortest time E such that a packet (w, s, F) is generated with probability $p_{\texttt{packet}}$ in time E, given a probability of entangled link generation $p_{\texttt{succ}}$.

a fixed rate, then the service model determines $R_{\texttt{attempt}}$. For example, if the service model specifies the requested rate is to be met exactly, then the central controller would set $R_{\texttt{attempt}} = R/p_{\texttt{packet}}$.

If the requested rate is 0, that is the nodes are requesting the minimal rate to achieve minimal service, then the central controller needs to calculate this rate using a method conforming to Algorithm 3. In particular, the central controller sets a value $\epsilon_{service}$ for the maximum probability that any session does not obtain minimal service. This parameter is known by both the central controller and the relevant end nodes. In one possible implementation, it may be set by the network operator when initially setting up the network. From this, the minimum number of PGAs needed to meet the service threshold can be determined. Once this number is known, the minimum rate can be calculated.

Algorithm 3: Determining the minimum rate of PGAs

Input : $N_{\texttt{inst}}, t_{\texttt{expiry}}$, current time, $\epsilon_{service}$, $p_{\texttt{packet}}$
Output: Minimum possible rate at which to schedule PGAs.

1 Calculate the minimum number of PGAs, $N_{\texttt{min}}$, required such that the probability of at least $N_{\texttt{inst}}$ packets being generated is at least $1 - \epsilon_{service}$;
2 **return** $N_{\texttt{min}}/(t_{\texttt{expiry}} - \texttt{current time})$

If permitted by the service model and any agreements

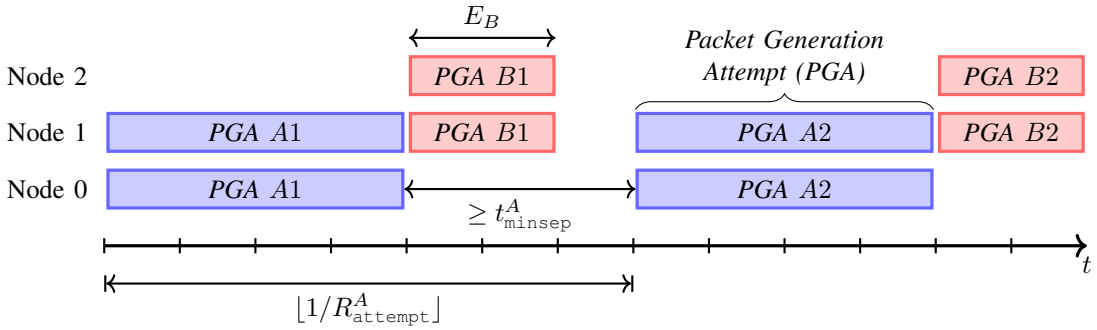


FIGURE 6: Excerpt from an example network schedule with two PGTs $\tau_A = ((4, 1/7, \{0, 1, M\}), 2, 100)$ and $\tau_B = ((2, 1/7, \{1, 2, M\}), 0, 100)$ on three nodes, where M represents a number of resources at a metropolitan hub.

with nodes, the central controller also retains the right to reduce, or throttle, the rates requested. This may be done in order to reduce the load on the network due to either a particular demand or the collection of all demands. Reducing the rate at which PGAs are scheduled may allow the network to serve more users at any one time, albeit at the cost of an increased probability that the throttled sessions do not obtain minimal service.

3) Additional Information

The scheduler requires knowledge of the resource requirements ρ of each demand. These depend on the path and entanglement generation protocol employed by the network to generate the desired entangled links, and can be obtained from the central controller's side of the network capabilities manager. In addition, the parameters governing the minimum separation time between subsequent PGAs, $t_{\texttt{minsep}}$, and the expiry time of the demand, $t_{\texttt{expiry}}$ are carried forward unchanged to the PGT. These final quantities are combined with the execution time of each PGA, E, and the rate at which PGAs should be scheduled, $R_{\texttt{attempt}}$, to define the complete PGT.

4) Complete Packet Generation Task

It is now possible to define the complete PGT.

Definition IV.10 (Realisation of a demand)**.** A *realisation* of a demand is a specific choice of how PGAs from a packet generation task will be executed. In particular, this includes specifying a protocol for generating end-to-end entangled links and which suitable packet will be generated. This also determines the path in the network along which links will be generated, and therefore the resources which will be required to execute each PGA.

We write the set of all possible realisations as $\mathcal{R}$.

Definition IV.11 (Packet Generation Task)**.** A *packet generation task* (PGT) is the following tuple:

$$\tau = \left(\left\{ (E, R_{\texttt{attempt}}, \rho)_r \right\}_{r \in \mathcal{R}},\ t_{\texttt{minsep}}, t_{\texttt{expiry}} \right) \quad (3)$$

where for each possible realisation $r \in \mathcal{R}$:

- E is the execution time of each PGA
- $R_{\texttt{attempt}}$ is the average rate at which PGAs are scheduled.
- ρ is the set of resources which are required to execute each PGA.

The relation of these parameters to a network schedule can be seen in Figure 6.

We make the distinction between demands, which pertain to the domain of the end nodes and PGTs which pertain to the domain of the central controller in order to maintain separability in the control structure. This separation reinforces that the central controller may determine how to realise a demand based on the information submitted and its knowledge about the state of the wider network.

If the network scheduling algorithm supports it, the packet generation task can be interpreted as a multi-mode task, similar to as in [84]. In such a situation, the network scheduler can choose the 'best' mode in which to schedule the packet generation task given the current network load. If such behaviour is not supported, then the central controller should reduce $\mathcal{R}$ to a single realisation/mode in which the PGT is always scheduled.

I. NETWORK SCHEDULING

Algorithm 4 summarises the procedure of network scheduling. The procedure is subdivided into the processes of scheduler admission control, computation of the schedule, and distribution of the schedule. Figure 4 illustrates possible timings for each of these processes.

1) Scheduler Admission Control

At the start of each scheduling interval, the first task the central controller undertakes is to identify which demands should be given service in the next network schedule. To do this, it will construct a set of PGTs for which PGAs will be included in the next network schedule.

Any PGTs from the previous schedule which have not expired or been terminated by the source nodes are auto-

Source	Supplied Parameters
Application script	$w, s, F, t_{\texttt{minsep}}$
End node(s)	$R, t_{\texttt{expiry}}, N_{\texttt{inst}}$
Central controller (configuration)	$p_{\texttt{packet}}, \varepsilon_{service}$
Central controller (calculation)	$E, R^{\texttt{attempt}}$
Path and protocol for entanglement generation	p_{succ}, ρ

TABLE 1: Table of entities (sources) and the parameters they provide.

Algorithm 4: Network Scheduling

Input : Demands in queue, previously scheduled PGTs, scheduling interval, service model

Output: Network Schedule

1 Decide which, if any, demands to accept from the queue into the schedule;
2 Compute the schedule for the next scheduling interval following the chosen service model;
3 Distribute the schedule to all relevant parties;

matically carried forward into this new set of PGTs. Furthermore, the admission control process may take demands from the demand queue, convert them into PGTs and then decide whether or not to add them into the set of PGTs to be scheduled. If the admission control algorithm decides to add a given PGT, we say the corresponding demand is accepted and exits the demand queue. If a PGT cannot be added (or the admission control algorithm decides not to add it), then the demand is not accepted, and we leave the precise behaviour in this case to an implementation. Examples of possible behaviours could range from leaving the demand in the queue and attempting to admit it in the next schedule, to rejecting the demand outright and removing it from the demand queue.

Any specific implementation of the architecture may define a tailored admission control routine which addresses the performance goals of the implementation. However, a guideline for any implementation is that the utilisation of any resource, $\mathcal{U}_r$, should satisfy

$$\mathcal{U}_r = \sum_{\tau : r \in \rho_\tau} U_\tau \leq 1,$$

where $U_\tau = E_{r,\tau} R_\tau^{\texttt{attempt}}$ and r is the specific realisation which is scheduled. This is to ensure the schedules are feasible.

2) Computing the Network Schedule

Once the set of PGTs to schedule has been finalised, the central controller constructs the network schedule for the next scheduling interval, using a method conforming to Algorithm 4. In doing so, each PGT which has been accepted for scheduling is assigned a series of start times, from each of which a PGA is executed without preemption.

Such a schedule may be based on fixed duration time-slots, though this is not required. If a fixed duration time-slotted schedule is employed, the central controller determines the length of the time-slots, which are the same for all resources in the network schedule.

The computed network schedule needs to cover the entirety of a scheduling interval. This can be achieved either by directly computing a schedule for the whole scheduling interval or by computing a shorter schedule that can be repeated to cover the whole scheduling interval.

A constraint applies to the time required to compute the network schedule, and thus restricts the selection of a specific scheduling algorithm. This computation time should be short enough so that a single scheduling interval covers computation and distribution of the schedule, followed by a final buffer time for the construction of local schedules. End nodes require the final buffer to construct their local schedules, which incorporate the network schedule.

It is the role of an implementation to specify handling of the situation where the schedule is not computed in time. An example of a policy which may be employed is to delay execution of the late computed schedule to the start of the following scheduling interval. Alternatively, if the central controller distributes the schedule as soon as possible then each network component could start executing the schedule from the time when it is received. Whichever policy is enforced, if there is a gap between the end of the previous schedule and the arrival of a new schedule at a network component, a guideline for implementation is to instruct all components to only begin executing new PGAs upon the arrival of the new schedule. This ensures proper synchronisation across all components.

J. DISTRIBUTING THE NETWORK SCHEDULE

Once the schedule has been computed, it must be distributed to all components of the network. Each component only receives the portion of the network schedule relevant to it. For example, an end node receives only its portion of the schedule, whereas a metropolitan hub receives the portion of the schedule concerning all nodes to which it is connected. To ensure compatibility with requirements of the network stack and the runtime application execution environment on end nodes, the format of the schedule must be such that the relevant end nodes receive the start and end times of each scheduled PGA together with an identifier of which demand the scheduled PGA pertains to. Various implementations are possible.

As indicated in Figure 4, sufficient time must be allotted for distribution of the network network schedule. The interval reserved for distribution should be such that the probability that any component does not receive the network

schedule is vanishingly small. However, in the case that a component does not receive the schedule on time, we expect that the affected components will continue to request the network schedule from the central controller. Furthermore, as with the case where the schedule is not computed on time, such components should not start any new PGAs until they receive the correct schedule.

Once a network schedule is received by an end node, it can compute its local schedule. This uses the network schedule to determine when submitted QC (entanglement generation) blocks are executed (interaction **F** in Figure 5). The remaining instruction blocks are then scheduled relative to these QC blocks [16], [85]. Once this process has been completed, the local and network schedules can be executed, without any extra interaction either between pairs of end nodes and between end nodes and any internal network components (interaction **G** in Figure 5).

V. IMPLEMENTATION

In order to be able to perform an evaluation in §VI, we provide an example implementation of the central controller in our architecture, focusing on the process of computing the network schedule. All the algorithms explicitly presented in this section are applicable to any network topology, with any applications being executed over the network. Therefore, this implementation can act as a baseline against which future implementations can be compared.

A. NETWORK SCHEDULE FORMAT

We will implement the network schedule as a synchronous time-slotted network schedule, with timeslots of fixed length $t_{timeslot}$ seconds, similar to [28]. In our implementation, the duration of each of these timeslots is chosen to be of a similar order of magnitude as the expected time to produce a single end-to-end entangled link. This allows us to make the simplifying assumption that at most one link is generated in a given timeslot of the schedule.

It will also be convenient to express certain quantities in terms of numbers of time slots, rather than in seconds. In particular, we will re-normalise the window w to be expressed as a number of timeslots, and similarly for the execution time E of PGTs. Furthermore, it will often be more convenient to work with the *period* of PGTs (defined as the reciprocal of the rate of packet generation attempts), expressed again as a number of timeslots. Therefore, to meet the required rate of $R_{\texttt{attempt}}$ PGAs per second, there must be a PGA scheduled every $T_{\texttt{attempt}}$ timeslots.

We calculate the period by

$$T_{\texttt{attempt}} = \left\lceil \frac{1}{R_{\texttt{attempt}} t_{timeslot}} \right\rceil_{\mathbb{N}}, \tag{4}$$

where we write $\lceil x \rceil_{\mathbb{N}}$ to be x rounded to the nearest positive integer. Note that as $R_{\texttt{attempt}}$ is the number of PGAs per second, we expect $R_{\texttt{attempt}} < 1$ and from the assumptions above, we expect $t_{timeslot} \ll 1$. Therefore, we expect that $T_{\texttt{attempt}} \gg 1$ for most PGTs.

B. SERVICE MODEL

For our implementation, the network operates under the following network model:

Network Service Model SM1 (No Throttling)**.** Under this model, demands are met exactly, that is PGAs are scheduled to on average meet exactly the rate requested by the nodes. In the case where the network is oversubscribed, then demands which cannot be accepted to be scheduled are delayed until there is space, or dropped if they can never be served.

Note that this service model requires some admission control for the scheduler, to decide which demands are accepted for scheduling and which are delayed. We give some examples of such rules in §V-G1.

This service model is particularly applicable to scenarios where nodes are relying on demands being met exactly. An example of such a scenario would be using QKD to underpin the availability and monitoring of secure critical infrastructure. In such a case, a demand being throttled could potentially result in loss of access to the infrastructure, due to not being able to generate keys quick enough. Examples of such schemes include [86]–[88].

C. NETWORK CAPABILITY UPDATE

The evaluation will be performed on a network with a star topology (Figure 7), as it removes any influence from routing or determining schemes for generating end-to-end entangled links. Consequently, we will not give implementations of such schemes. For examples of how routing may be performed in more complicated network topologies, see for example [89] and [90].

D. CAPABILITY NEGOTIATION

We assume that capability negotiation is carried out using the *exposed hardware interface (EHI)* from Qoala [16]. This allows the end nodes to exchange information about the hardware and software constraints of their devices.

E. DEMAND REGISTRATION AND QUEUING

We use the following set of rules for demand registration:

Demand Registration Rule DR1. The demand must be sane. In particular we require $w \geq s$, where w is expressed as a number of timeslots, and that both parties are capable of generating entangled links.

Demand Registration Rule DR2. Let the utilisation of a PGT τ with execution time E and period $T_{\texttt{attempt}}$ be $U_\tau = E/T_{\texttt{attempt}}$. Then the utilisation of a PGT resulting from at least one of the requested service options must be less than $\hat{U} = 0.8$ (otherwise the demand would never be accepted by our admission control rules).

Once demands are registered, they are placed into a *first-in-first-out* (FIFO) queue for consideration by the scheduler admission control.

F. CREATING PACKET GENERATION TASKS

We use a fixed value for the probability, $p_{\texttt{packet}}$, that a packet is successfully generated by a PGA. Then, to determine the length of a PGA, we use results from *scan statistics*. Specifically we use the approximations in [26] (also given in Appendix C1) for the probability of k successes in a window w given a total of N trials. From this we can calculate the minimum execution time of the PGA, such that the PGA succeeds with probability $p_{\texttt{packet}}$, using interval bisection.

If the nodes have requested the central controller calculate the minimum rate ($R = 0$), we use Hoeffding's inequality to calculate the minimal N_{min} such that after N_{min} PGAs, the session obtains minimal service with probability $1 - \epsilon_{service}$. The rate of packet generation attempts is then set by $N_{min}/(t_{\texttt{expiry}} - \texttt{[current time]})$. This process is described in detail in Appendix C2. Otherwise $(R > 0)$, we simply set $R_{\texttt{attempt}} = p_{\texttt{packet}}^{-1} R$

G. NETWORK SCHEDULING

1) Admission of New Demands

The central controller decides which demands to remove from the queue by checking if accepting a demand would violate either of the following rules:

Scheduler Admission Control Rule SAC1 (Utilisation Bound)**.** For all resources r, the utilisation of r, $\mathcal{U}_r$, must satisfy

$$\mathcal{U}_r = \sum_{\tau : r \in \rho_\tau} U_\tau \leq \hat{\mathcal{U}} \tag{5}$$

for some constant $\hat{\mathcal{U}} \in (0, 1]$.

Scheduler Admission Control Rule SAC2 (Computation Time Bound)**.** The estimated time to compute the network schedule cannot exceed $\alpha_C T_{SI}$, for some constant $\alpha_C \in (0, 1)$.

If neither rule is violated, then the demand is accepted and the corresponding PGT created. Otherwise, unless DR2 is violated, the demand is returned to the head of the queue. Since the queue is *FIFO*, demands can only be accepted in the order they were submitted. If a given demand fails admission control but is not removed from the demand queue, then no more demands can be accepted from the demand queue until the next scheduling interval.

The utilization bound $\hat{\mathcal{U}}$ in SAC1 is restricted to $(0, 1]$ as if any $\mathcal{U}_r > 1$, then there is not physically enough time to schedule all the required PGAs. The estimated computation time in SAC2 may be continuously updated by the central controller based on its current performance when computing network schedules. We cannot allow $\alpha_C \geq 1$, as there must be some time remaining in the scheduling interval to distribute the schedule, as per Figure 4.

We write $\mathcal{T}$ to be the set of PGTs for which PGAs must be scheduled in the next network schedule. Every PGT in $\mathcal{T}$ corresponds to a demand that has passed admission control and has neither expired nor been terminated. Once a demand has been terminated or expires, then no more PGAs will be scheduled from it.

2) Computing the Network Schedule

To calculate the network schedule, we adapt priority-based periodic task scheduling methods from real time systems, and use a synchronous time-slotted network schedule with time-slots of fixed duration $t_{timeslot}$, similar to as in [28]. In particular, we will use *earliest deadline first (EDF)* scheduling, and adapt the following model, which is common to real-time periodic scheduling (*c.f.*, [91], [92]): *A periodic task creates jobs. Each of these jobs is released at some time r, only after which can it be executed, and must be completed by a deadline d. If a job is the ith released by a task, then its release time is given by $r = (i-1)T + \sigma$ where T is the period of the task and σ is an offset determining how long after the start of the schedule the first job is released. The corresponding deadline is then set to the start of the next period, i.e. $d = r + T = iT + \sigma$.*

In our case, the tasks are PGTs and the jobs are PGAs. Following the definitions in [91], we take our deadlines to be *soft*, which permits PGAs to be scheduled past their deadlines if it is not possible to do otherwise. In this way, no PGAs will be skipped, and so the average rate of packet generation experienced by the end nodes will be as they requested over the course of the entire lifetime of the demand.

To be able to determine the release times and deadlines of each PGA, the period of the PGT, as given by (4), is required. We also need to determine the offset σ_i. This is set to be the start of the scheduling interval during which PGT τ_i is first scheduled. We can now determine the release time and deadlines of each PGA which needs to be scheduled.

For $\tau_i \in \mathcal{T}$, let $\tau_{i,j}$ be the jth PGA from PGT i. Let T_i and σ_i be the period and offset of τ_i respectively, let $r_{i,j}$, $d_{i,j}$ be the release time and deadline of $\tau_{i,j}$, and let $s_{i,j}$ and $c_{i,j}$ be the start and completion times of $\tau_{i,j}$, respectively. Then,

$$r_{i,j} = \sigma_i + \max\{(j-1)T_i, c_{i,j-1} + t_{\texttt{minsep,i}}\} \tag{6}$$

$$d_{i,j} = \sigma_i + jT_i \tag{7}$$

where $j = 1, ..., \lfloor (t_{\texttt{expiry},i} - \sigma_i)/T_i \rfloor$. Note that we adapt the determination of the release times from the usual formula in order to incorporate the minimum separation time between two PGAs.

Definition V.1 (Eligibility for scheduling of a PGA)**.** We say that a PGA $\tau_{i,j}$ is *eligible* for scheduling at time t^* if it meets the following criteria:

1) It has been released ($t^* > r_{i,j}$)
2) It has not yet been scheduled (the start time $s_{i,j}$ is undefined)
3) The required resources are available

The schedule is then computed according to Algorithm 5.

Algorithm 5: EDF Scheduling Algorithm.

Input : Desired schedule length, PGTs to be scheduled, $\mathcal{T}$
Output: Scheduled PGAs

```
Set the initial decision time t* to the time at the start of the schedule;
while t* is less than the end time of the schedule do
    while there are eligible PGAs do
        Schedule the eligible PGA with the earliest deadline;
        Update the set of eligible PGAs;
    end
    Update t* to the next time either a task is released, a task completes execution, or the end of the scheduling interval, whichever is earlier;
end
```

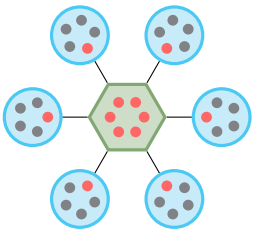


FIGURE 7: Example of a star-topology network with 6 nodes. The outer circles represent the end nodes and the central hexagon a central junction node. In each network time-slot the central node can attempt to create an entangled link with each of the end nodes, and then perform entanglement swaps to create end-to-end links between pairs of end nodes. The orange dots represent communication qubits and the black dots represent memory qubits.

H. END NODE SCHEDULING

As our evaluation focuses on the network portion of the architecture, we do not compute the local schedules. We assume that node schedules exist which facilitate the successful execution of an application instance given that packet generation succeeds. For examples of methods of constructing schedules on end nodes, see the scheduler built into the Qoala environment simulation in [16] or one of the schedulers used in [15], [85].

VI. EVALUATION

We now evaluate the performance of our network architecture using the implementation described in the previous section. To do so, we simulate two test applications from different application classes. We define the performance metrics we consider, and where relevant we provide further details about the precise model used in our simulations. Further details about the simulation model can be found in Appendix D. From the results we obtain we validate the viability of our architecture and are able to draw conclusions about the need for good admission control and how nodes should decide what rates of packet generation to request.

A. METRICS

Definition VI.1 (Termination of a Session)**.** We say a session is *terminated* if the session has obtained minimal service and sent a termination message to the central controller. This results in no further time being allocated for this demand beyond the end of the schedule currently being computed.

Definition VI.2 (Expiry of a session)**.** We say a session is *expired* if the expiry time has elapsed without a termination message being received. No packet generation attempts will be scheduled after the expiry time.

Note that a session can expire and still have obtained minimal service, if the nodes choose not to send the termination message and carry on generating packets. Likewise, a node could choose to send a termination message before obtaining minimal service. However, we assume in our evaluation that neither of these behaviours occur.

Metric I (Proportion of Expired or Terminated Sessions Obtaining Minimal Service)**.** Let $\mathbb{S}$ be the set of application sessions initiated in a given simulation. Let $\hat{\mathbb{S}} \subseteq \mathbb{S}$ be the set of those application sessions which ultimately expire or are terminated. Let $\mathcal{M} \subseteq \hat{\mathbb{S}}$ be the set of those application sessions which obtained minimal service. Then the metric, p^{MS}, is given by

$$p^{MS} = \frac{|\mathcal{M}|}{|\hat{\mathbb{S}}|} \in [0, 1]. \tag{8}$$

Such a metric is important to both the end nodes and the central controller. End nodes can use this to estimate the likelihood of a submitted application session obtaining minimal service, and therefore what quality of service the network can provide. On the other hand, by monitoring this metric, the central controller can assess the effectiveness of the (scheduler) admission control rules given the traffic on the network, and update them accordingly.

Metric II (Average Time Spent in Queue)**.** Let $t_{submit,\mathcal{S}}$ be the time at which the demand corresponding to session $\mathcal{S}$ is submitted to the central controller. Let $t_{exit,\mathcal{S}}$ be the time at which the demand corresponding to session $\mathcal{S}$ leaves the queue. Then the metric is given by

$$\bar{t}_{queue} = \frac{1}{|\mathbb{S}|} \sum_{\mathcal{S} \in \mathbb{S}} \Big(t_{exit,\mathcal{S}} - t_{submit,\mathcal{S}} \Big). \tag{9}$$

For end nodes, evaluating this metric gives them an estimation of the expected latency between submitting a demand and receiving service. This in turn can then be used to estimate the load on the network, and even inform the choice of expiry time for the application session. For the central controller, this metric gives an estimation of how overloaded the network is, especially in conjunction with p^{MS}. Using this, the central controller can control the

Parameter	Value
Network time-slot length	$100\,\mu$s
p_{gen}	7.5×10^{-5}
F	0.925
qubits at end node	5

TABLE 2: Network Parameters for Evaluation. p_{gen} is the probability of an end-to-end entangled link being created in a given time-slot. The trends we observe are generally insensitive to the value of p_{gen}.

Parameter	Value
$p_{\texttt{packet}}$	0.2
$\epsilon_{service}$	10^{-5}
PGA cap per schedule	1500
Utilisation Bound per link, $\hat{\mathcal{U}}$	0.85
Computation time factor, α_C	0.5
Scheduling Interval (MDA|CKA)	300s | 3600s

TABLE 3: Scheduler Parameters for Evaluation. Except where noted, the trends that we observe are insensitive to the specific values chosen here.

overload by either adapting the admission control rules as required, or even requesting that nodes reduce the rate at which new demands are submitted.

B. NETWORK CONFIGURATION

We consider a network consisting of 6 end nodes connected to a single central node as shown in Figure 7. Each end node is equipped with 5 qubits, of which at most one can be used for entanglement generation at any given time. The central node also has 6 independent communication qubits with no storage capability beyond the current network time slot. We assume that in every time slot, an elementary entangled link may be generated between each end node and the central node.

Links with the central hub are consumed by performing deterministic entanglement swapping operations to generate end-to-end entangled links between pairs of end nodes. The swap operations which are carried out are determined by which pairs of end nodes have a PGA scheduled in that time slot. We assume that the central node has no quantum memory and therefore any link which is generated must be consumed within the same time slot or be lost.

We therefore simulate a mathematically equivalent model, whereby for each disjoint pair of nodes, an end-to-end link may be generated in each time slot with fixed probability $p_{\text{gen}} \ll 1$.

C. ADMISSION CONTROL

1) Utilisation Bound

We set the value of the utilisation bound $\hat{\mathcal{U}} = 0.85$. A restriction to $\hat{\mathcal{U}} \leq 1$ is necessary to ensure the schedule is feasible. However, as the tasks are non-preemptable, this is not a sufficient condition because of so-called priority inversions. These occur when a task is available to be scheduled, but there is a non-preemptable lower priority task which is currently being executed preventing the higher priority task from being scheduled [92]. As it is non-trivial to determine if a deadline will be missed without computing the schedule [79], we instead reduce the utilisation bound to reduce the probability that a priority inversion causes a missed deadline. By performing additional simulations beyond those reported on here, we observed that the conclusions we draw hold for any value of $\hat{\mathcal{U}} < 1$.

2) Schedule Computation Time

We set $\alpha_C = 0.5$, i.e. network schedules cannot take more than half the scheduling interval to compute. However, to avoid our results depending on the hardware of the server on which we perform our simulations, we do not directly implement SAC2. As the time to compute the schedule depends on the number of PGAs which will need to be scheduled, the central controller can implement this rule by imposing a cap on the estimated number of PGAs which need scheduling in a given scheduling period. Therefore, we simply fix this cap *a priori* and do not update it based on the observed computation times.

The central controller will be running on dedicated hardware in any actual deployment. This means that we do not expect the computation times to fluctuate much as there will not be other background processes consuming CPU resources. Therefore, despite fixing this cap on the number of PGAs per schedule to be constant, the results obtained will still be indicative of real-world performance. In our simulations, we set this cap to 1500 PGAs per schedule. This is motivated by empirical characterisation of the time our computation server takes to calculate the network schedule for various numbers of PGAs, more details of which can be found in Appendix D3.

D. SESSION MODEL

1) Creation time

Let $t_{\mathcal{S}}^{MS/E}$ be the time at the end of the scheduling interval during which application session $\mathcal{S}$ either expires or obtains minimal service, and let t_{renew} be an exponentially distributed waiting time with parameter λ. Then the nodes involved in $\mathcal{S}$ begin a new session $\mathcal{S}'$ of the same application at time $t_{\mathcal{S}}^{MS/E} + t_{renew}$. In each simulation, the value of λ is the same for all pairs of nodes.

2) Contents

In each of our simulations, all pairs of nodes execute the same application sessions. Two test applications are considered: the first is a measure-directly application motivated by QKD (hereafter MDA), and the second is a create-and-keep application motivated by the blind quantum computing algorithm in [83] (hereafter CKA). For further discussion of the distinction between measure-directly and create-and-keep application classes refer to Section III-B2. The Qoala files we use for these test applications are given in Appendix D6. We fill out the remaining fields in the

application sessions as follows:

$$\mathcal{S} = (\mathcal{N}, \texttt{APP}, N_{\texttt{inst}} = 100, \\ t_{\texttt{expiry}} = t_{submit} + t_{\texttt{max duration}}) \quad (10)$$

where $\texttt{APP} \in \{\texttt{MDA}, \texttt{CKA}\}$, t_{submit} is the time the corresponding demand is submitted to the central controller and $t_{\texttt{max duration}}$ is the maximum duration of a session.

We acknowledge that the value of $N_{\texttt{inst}} = 100$ is very low, as for example a typical QKD application would require $N_{\texttt{inst}} \gg 10^5$. However, we make this choice for simulation purposes, as it cuts down the required run time of our simulations (some of which run in almost real-time). We expect that the conclusions drawn from our simulation results remain valid when $N_{\texttt{inst}}$ is increased to values which more accurately reflect the requirements of real-life applications.

3) Peer-to-Peer vs Client-Server

We consider two regimes under which pairs of nodes execute sessions, *peer-to-peer* and *client-server*. In the peer-to-peer regime, an end node can undertake an application session with any other end node. This is the type of behaviour we would expect for applications such as QKD, where every end node in the network is capable of carrying out the application. In the client-server regime, one of the nodes is designated as the 'server', and the rest as 'clients'. Client nodes are only able to undertake sessions with the server and not with each other. This type of behaviour we would expect for applications such as BQC, where one end node needs to be much more powerful than the other. We use the first regime for modelling MDA traffic and the second for CKA traffic.

4) Termination of Demands

We assume that nodes will send a message to the central controller to terminate their demand as soon as they obtain minimal service. Once a demand is terminated, it is not considered further for scheduling. Nodes will, however, continue to execute application instances whilst there is time allocated in the network schedule.

E. RESULTS

Simulation of the implementation (Section V) of our network architecture is an opportunity to confirm the feasibility of our architecture and to asses the performance of an implementation, as quantified by the metrics in Section VI-A. These metrics may depend on parameters that are specific to how users demand service from the network as well as on the types of applications from which user demands originate. We focus on assessing how changes to the parameters λ and R, respectively the session renewal rate and the requested rate of packet generation, impact the values obtained for the performance metrics. To asses the impact of the type of application, we also compare the results of simulations where all nodes are running the MDA test application in the peer-to-peer regime (Figure 8) to simulations where all nodes are running the CKA application in the client-server regime (Figure 9).

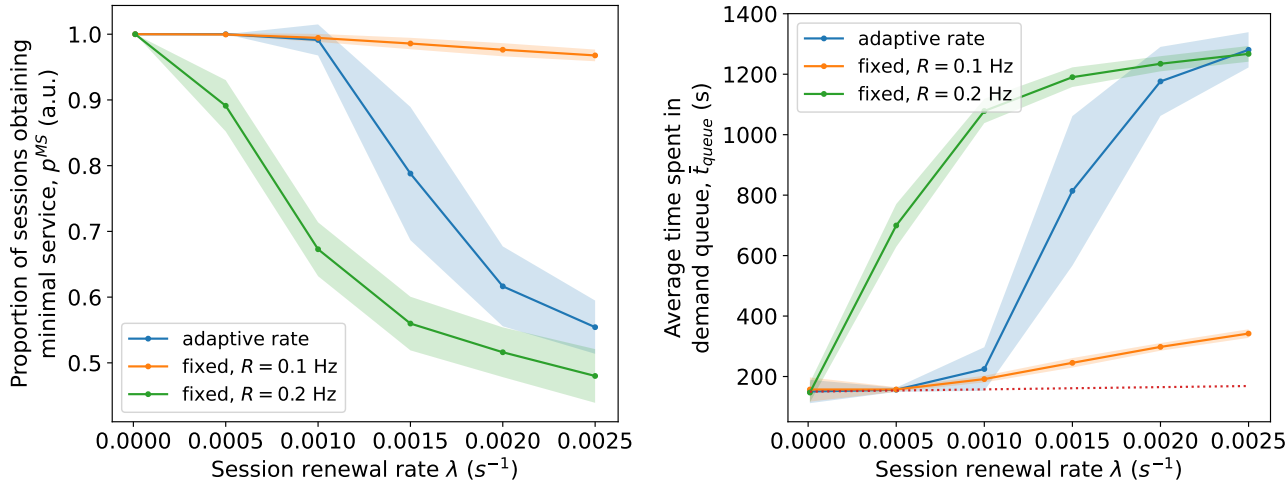
1) Facilitation of obtaining minimal service

The most important criteria that a network architecture must meet in order to be considered viable is the capability to meet user demands. Here we quantify the degree to which an implementation of our architecture meets user demands with the performance metric *proportion of expired or terminated sessions which obtain minimal service*, p^{MS}, quantified by (8). For each of the test applications simulated we observe that the implementation of our network architecture is able to successfully deliver minimal service to some proportion of sessions. For the MDA test application operated in the peer-to-peer regime (Figure 8a) the proportion of demands which obtain minimal service is always at least 0.45, for all parameter combinations simulated. In complement, we observe that for the CKA test application operated in the client-server regime a much higher proportion (>0.98) of sessions obtain minimal service, for each combination of the parameters simulated.

This general increase in the value of p^{MS} between the client-server CKA and peer-to-peer MDA simulations can be attributed in part to the decrease in the number of possible demands. In the client server regime, there are only 5 sources of demands, whereas in the peer-to-peer regime there are 15 possible sources of demands. As each source only has a single demand registered at any time, the maximum length of the queue is much shorter in the client-server CKA simulations. This in turn means that as a fraction of the value of $t_{\texttt{max duration}}$, the average queuing times are much shorter in the CKA simulations, leading to greater values of p^{MS}.

From Figures 8a and 9a it is clear that the session renewal rate λ and the requested rate of packet generation R have a large impact on the proportion of sessions which obtain minimal service. In particular, as λ increases, the proportion of sessions which obtain minimal service may only decrease. This is the expected behaviour, as an increase in λ directly translates into increased load on the network. We do note that in the case where $R = 0.001$Hz in Figure 9a, $p^{MS} = 1$ for all λ, with only a small non-zero standard deviation for $\lambda > 2 \times 10^{-5}$. This indicates that there are parameter regimes where it is possible for all nodes to obtain minimal service, regardless of the session renewal rate. Figures 8b and 9b confirm that the average time a demand spends in the queue, $\bar{t}_{queue}$ always increases as λ increases. An extension of the duration of time that demands spend queued is direct evidence of increased load on the network.

The effect of R is more subtle, as we consider two values of a fixed rate as well as an adaptive rate. In the case of the MDA test application operated in peer-to-peer mode (Figure 8) the lower fixed rate results in higher p^{MS} for all values of λ. In contrast, for the CKA test application operated in client-server mode, the lower fixed rate results



(a) (b)

FIGURE 8: Results from simulations for peer-to-peer MDA on a six-node star network. (a) Proportion of initiated sessions which obtained minimal service from the network. (b) The average time a demand spent in the demand queue. The shaded area represents $\pm 1\sigma$. The red dotted line represents the expected value of $\bar{t}_{queue}$ for a single pair of end nodes submitting demands. The total simulated time was 6 hours. To produce each data point we average the results of 100 simulations, with simulations where zero sessions were initiated removed. There were no such simulation runs observed.

in lower p^{MS}. See §VI-E4 for an explanation for why this is the case. However, as could be expected, for both test-applications the lower fixed rate results in lower $\bar{t}_{queue}$. This is expected because a lower fixed rate again translates to lower load on the network. In section §VI-E3, we comment on the effectiveness of adaptive rate requests, in comparison to requests for fixed rates.

2) Admission Control Requirements

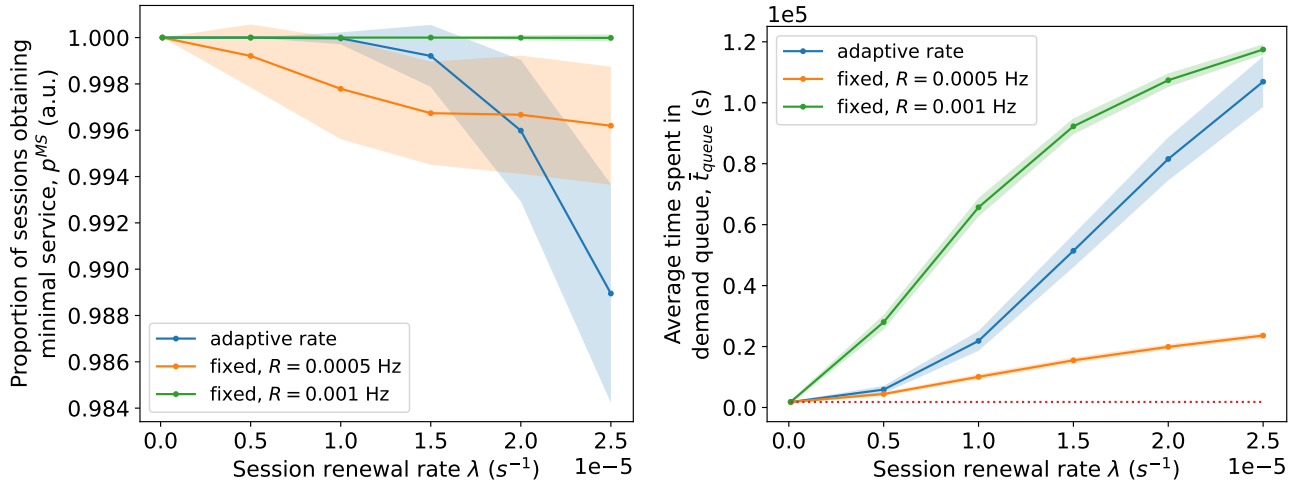
Whilst being able to facilitate some application sessions obtaining minimal service is sufficient for our architecture to be viable, it is desirable that this is the case for as many sessions as possible. In the peer-to-peer MDA simulations, however, we observe that as λ increases, the value of p^{MS} drops below 0.6 for the adaptive rate demands and below 0.5 for the fixed $R = 0.2$Hz demands. This is due to the increased load which these demands place on the network, compared to the demands with a fixed rate of $R = 0.1$Hz. In particular, as each demand/PGT requires a greater utilisation of the links to the central junction node, fewer demands can be served at any one time. This leads to increased queuing times, as observed in Figure 8b. Therefore, a greater number of demands reach their expiry time without getting scheduled, and moreover those demands which are accepted for scheduling have much less time for generating packets, leading to a lower probability of obtaining minimal service.

To prevent the value of p^{MS} from dropping so far, the central controller should not allow the network to become so overloaded. This may be achieved through the admission control rules employed, both at demand registration and scheduler admission. In particular, we expect that if a demand is rejected, for example for requesting too high of a rate of packet generation, then the nodes would re-submit this demand with, for example, lower values of R. From the data we observed, we can conclude that the demand registration rules in our implementation were not stringent enough for the peer-to-peer MDA scenario. In particular, the fixed rate $R = 0.2$Hz demands probably should almost always have been rejected, as accepting them led to a situation where a significant proportion of sessions were not able to obtain minimal service from the network (up to 50% by $\lambda = 0.0025$). However, the same set of admission control rules was sufficient for the client-server CKA scenario ($p^{MS} > 0.98$ for all λ). Therefore, when designing the set of admission control rules to implement, not only should the properties of the demands themselves be taken into account, but also the frequency and number of demands which the central controller expects to receive.

3) Controls on Requesting Adaptive Rates

When nodes submit a demand to the network, they may either request a fixed or adaptive rate of packet generation. One may expect that requesting an adaptive rate of packets is beneficial, as any time that the demand spends queuing is accounted for when determining how frequently PGAs need to be scheduled. For adaptive rates therefore, the process of queuing should not affect the probability that a session obtains minimal service, and the value of p^{MS} for adaptive rate demands should be greater than that of fixed rate demands for all λ.

In both the peer-to-peer MDA and the client-server CKA simulations, this expected behaviour is observed for small



(a) (b)

FIGURE 9: Results from simulations for client-server CKA on a six-node star network. (a) Proportion of initiated sessions which obtained minimal service from the network. (b) The average time a demand spent in the demand queue. The shaded region represents $\pm 1\sigma$. The red dotted line represents the expected value of $\bar{t}_{queue}$ for a single pair of end nodes submitting demands. The total simulated time was 360 days. All data points are the average of 250 simulations, with simulations where zero sessions were initiated removed. There were no such simulation runs observed.

values of λ, with the value of p^{MS} matching the value of the best fixed rate demands. However, once the value of λ passes a critical value (0.001 for MDA and 10^{-5} for CKA), the value of p^{MS} obtained by the adaptive rate demands is less than that of at least one of the fixed rate demands.

This reversal can be explained by considering the effect that queuing has on the rate at which PGAs are scheduled, $R_{\texttt{attempt}}$, for adaptive rate demands. Recall that when end nodes request an adaptive rate, the central controller will attempt to schedule at least N_{min} (equal to 850 for both test applications) PGAs in the time before the demand expires. This ensures that if a demand is accepted for scheduling, the session will obtain minimal service with probability at least $1 - \epsilon_{service}$. The consequence of this is that the longer an adaptive rate demand waits in the demand queue, the greater the resulting value of $R_{\texttt{attempt}}$ will be. This in turn leads to each PGT contributing more PGAs to a schedule, and utilising more of the resources in the network. Therefore, fewer PGTs can be scheduled in any given network schedule, leading to even longer queuing times and creating a feedback loop. In particular, more demands will expire before being scheduled, leading to the observed decrease in the value of p^{MS}. This queuing behaviour is also reflected in Figures 8b and 9b, where a (sharp) increase in the value of $\bar{t}_{queue}$ occurs as λ passes the critical values above.

4) Best Fixed Rate for Nodes to Request

End nodes may alternatively derive some benefits by requesting a fixed rate of packet generation. For instance, the rate of packet generation will be independent of any other network traffic, and can be chosen to reflect other requirements of the application not captured by, say the expiry time. For example, suppose that Alice and Bob want to generate a QKD key within the next half hour, but once they start generating the key, they want to complete key generation within a minute. This desired behaviour could be captured by requesting a high fixed rate of generation, substantially larger than the minimum rate to generate the required packets across the full half hour before the application session expires. Furthermore, as we have seen in the previous section, there are even some network conditions where requesting a fixed rate will lead to a greater likelihood of obtaining minimal service from the network.

From the results of the peer-to-peer MDA simulations in Figure 8, we observe that it is beneficial for end nodes to request a lower rate of packet generation. This not only leads to an increased value of p^{MS}, but also shorter queuing times and latency before being scheduled. In comparison, in the client-server CKA simulations, we observe the opposite effect. Whilst increasing the requested rate of packets does increase the queuing time, the value of p^{MS} also increases (it is exactly 1 for all λ).

We can explain this behaviour as follows: In general it is beneficial to request a higher rate of packet generation. Requesting a higher rate of packet generation will result in more PGAs being scheduled in any given time interval, increasing the likelihood of obtaining minimal service. In particular, this also makes them more resilient against having to wait in the demand queue.

However, higher rates also results in longer queuing times through greater utilisation and PGA contributions to a sched-

ule. Therefore, there is a critical value of R, depending on the demands and demand submission characteristics, where the effects from extended queuing times starts to outweigh the benefits of potentially more PGAs being scheduled. This is precisely what we see in Figure 8a with the fixed $R = 0.2$Hz demands for peer-to-peer MDA. On the other hand, for the client-server CKA, $R = 0.001$Hz is still below this critical value of R, and so the demands gain all the benefits of scheduling PGAs at a higher rate. The existence of such a critical value of R is not surprising, as networks with control of limited numbers of resources are well known to have finite capacity regions (e.g. [65], [93]).

VII. CONCLUSIONS AND FUTURE WORK

We have designed a novel architecture for a quantum network which allows for the integration of network scheduling with local program execution. This is achieved by introducing an application-motivated demand format for end nodes to request packets of entanglement generation. This is in contrast to more limited demand formats in previous work. In such demands, one requests a rate of end-to-end entangled link generation, which fails to capture some of the requirements for executing applications.

We also defined a scheme for producing network schedules in a format which can be used by the local execution environment on an end node to more efficiently schedule local operations. To do this, we introduced packet generation tasks and packet generation attempts to allow the central controller of the quantum network to effectively allocate the use of shared resources in order to satisfy submitted demands.

We presented an example implementation of our architecture using a network scheduler based on EDF scheduling. Using this implementation, we performed numerical simulations of our architecture on a star-shaped network. The results of these numerical simulations highlight that the architecture successfully provides application sessions with minimal service, both for measure-directly and create-and-keep type applications. Furthermore, we have seen that there is a need for smart admission control, both in selecting which demands are accepted upon arrival and in selecting which queued demands to accept for scheduling. Finally, according to the performance metric of the proportion of sessions obtaining minimal service, we have seen that there is no single best fixed rate to request, but rather the optimal rate depends on current traffic on the network.

Our contributions open up many opportunities for future research. Each of the phases of the architecture, from admission control, to queuing, to computing the network schedule, requires a tailored algorithm. Design and optimisation of various possibilities for each type of algorithm merits in depth investigation. The properties of such algorithms directly impact the performance of the network. In particular, the choice of scheduling algorithm will have a great effect on the specific performance guarantees which the network can promise to the nodes.

We have shown that there is a need for efficient admission control. An algorithm for admission control should be tailored to enforce the desired network behaviour and to account for the expected traffic on the network. Given a target set of performance metrics, the impact of an admission control algorithm depends strongly on the scheduling algorithm. Thus, we expect scheduling algorithm design to inform the design of admission control algorithms, perhaps following some of the same ideas as in [94].

We also showed that whilst adaptive rates may seem like a good idea, they require extra care to handle to avoid the unintended consequence of leading to overload on the network. Whilst such requests are good at adapting to momentary variations in the load on the network, if the increased demand is sustained then the benefits are lost. This is consistent with previous work on rate control algorithms in both the classical and quantum domain, which suggests that for variable request rates to be effective they need to be combined with a rate control algorithm (see for example [65], [93]). It remains to determine conditions on when precisely this occurs, and bounds on how much the value of $R_{\texttt{attempt}}$ may vary without causing an overload.

Similarly, the optimal fixed rate which nodes should request merits further investigation. We have shown here that in some cases it is better to request a higher fixed rate, and in other cases it is better to request a lower fixed rate. However, it remains to determine precisely what is causing this bifurcation, and what the optimal rate to request is.

Finally, whilst we have only considered bipartite applications in our implementation and analysis, there do exist quantum network applications involving multipartite entangled states, for example in the domain of anonymous transmission [95], [96]. The architecture we have presented can in principle support multi-partite applications, however to enable this, one must first define a new notion of a 'packet of entanglement' to encompass more complex entangled links, and also define a function which can convert demands for these packets into PGTs. Given both of these, the rest of the architecture should then operate as described above, replacing the functions we define for bipartite applications and demands with those for multipartite demands. We leave the development of such packet formats and conversion functions to future work.

APPENDIX

A. NOTATION

A summary of the notation used in the paper is given in Table 4.

B. WHO KNOWS WHAT

1) Central Controller

The central controller *knows* the following information:

- Probability of PGA success p_{packet}
- Network topology
- Network traffic

Symbol	*Definition*
	Application Sessions
$\mathcal{S}$	Application session
$\mathcal{N}$	Set of nodes in the network
$\texttt{App}$	placeholder for a quantum application
$N_{\texttt{inst}}$	Minimum number of application instances required for minimal service
$t_{\texttt{expiry}}$	Expiry time of an application session/demand/packet generation task
	Packets of Entanglement
p	packet of entanglement
w	time window of packet
s	number of pairs in a packet
F	(average) minimum fidelity of links generated as part of a packet
	Demands
$\mathcal{D}$	Network Demand
R	Requested rate of packet generation
$t_{\texttt{minsep}}$	Minimum time between two packet generation attempts
	Packet Generation Tasks
τ	Packet generation task (PGT)
$\tau_{i,j}$	The jth packet generation attempt (PGA) stemming from PGT τ
E	Execution time of a PGA
$R^{\texttt{attempt}}$	(Minimum) rate at which PGAs should be scheduled
ρ	Resources required to execute a PGA.
	Network Schedules
T_{SI}	Duration of the scheduling interval
	Evaluation
$\epsilon_{service}$	Probability that a session does **not** obtain minimal service
$p_{\texttt{packet}}$	Probability that a packet is generated in a given PGA
p_{gen}	Probability that an attempt to create a single entangled link succeeds
$t_{\texttt{submit}}$	Time at which a demand is received by the central controller
$t_{\texttt{max duration}}$	Maximum duration of an application session. Used for dynamically setting expiry times.
	Applications
$\texttt{MDA}$	Measure Directly Application, based on QKD
$\texttt{CKA}$	Create and Keep Application, based on BQC
	EDF Scheduler
T	(Effective) period of a PGT
σ	Offset from $t = 0$ of a PGT
$r_{i,j}$	release time of PGA j from PGT i
$d_{i,j}$	deadline of PGA j from PGT i
$s_{i,j}$	start time of PGA j from PGT i
$c_{i,j}$	completion time of PGA j from PGT i
t^*	scheduler decision time
	Admission Control
U_τ	Utilisation of PGT τ
$\mathcal{U}_r$	utilisation of resource r
$\hat{U}$	Utilisation bound
α_C	Proportion of the scheduling interval allocated to compute the network schedule within.
$\mathcal{T}$	Set of PGTs from which PGAs should be scheduled in the next network schedule.

TABLE 4: Notation used throughout this paper. Entries which fall into multiple categories are only shown once.

- EGPs available on the network
- Capabilities of internal components, e.g. repeater chains.

The central controller *does not know* the following information:

- If a PGA is/was successful
- What application(s) are being run

2) End Nodes

The end nodes in the network *know* the following information:

- Local hardware capabilities
- Local utilisation
- Application program(s) [Number of pairs required, local gates]
- If an application instance executes (successfully)
- Identity of nearest neighbour and metropolitan hub.

The end nodes of the network *do not know* the following information:

- Full network topology
- EGPs used on the network
- p_{packet} (up to empirical deduction)
- Application sessions being run on other nodes.

C. METHODS FOR CREATING PACKET GENERATION TASKS

1) Approximations from Naus '82 for determining the length of PGAs

When determining how long a PGA should be, we need to be able to calculate the probability of a packet of entanglement being generated in some timeframe consisting of a known number of trials. This is precisely the sort of

problem which is addressed in the field of *scan statistics*, which is dedicated to looking at the probability that random events are grouped together [27]. In particular, the process of generating a packet is equivalent to the *generalised birthday problem*, which looks at the probability of getting k events (or successes) in a window of size m. Exact formulae as well as approximations exist to calculate these probabilities, we use the one in the following section which is due to Naus [26].

The following theorem and discussion is due to Naus [26]:

Theorem 1. *Let $T_{k,m}$ be the time at which we first observe k events in a window of size m. Then if we write*

$$\begin{aligned} P'(k|m,N,p) &= \mathbb{P}[T_{k,m} < N] \\ &= 1 - Q'(k|m;N;p), \end{aligned}$$

and abbreviate $Q'(k|m;Lm;p)$ as Q'_L, then

$$Q'_L \approx Q'_2 \left(\frac{Q'_3}{Q'_2}\right)^{\frac{N}{m}-2} \tag{11}$$

Naus also gives formulae for calculating Q'_2 and Q'_3 exactly:

Let

$$b(k;m,p) = \binom{m}{k} p^k (1-p)^{m-k},$$

$$F_b(r;s,p) = \begin{cases} \sum_{i=0}^{r} b(i;s;p) & r = 0,1,...,s \\ 0 & r < 0 \end{cases}.$$

Then for $2 < k < N$, $0 < p < 1$, we have

$$\begin{aligned} Q'_2 = &\left(F_p(k-1;m,p)\right)^2 \\ &- (k-1)b(k;m,p)F_b(k-2;m,p) \\ &+ mpb(k;m,p)F_b(k-3;m-1,p) \end{aligned} \tag{12}$$

and

$$Q'_3 = \left(F_b(k-1,m,p)\right)^3 - A_1 + A_2 + A_3 - A_4 \tag{13}$$

where

$$\begin{aligned} A_1 = 2b(k;m,p)F_b(k-1;m,p)\Big\{&(k-1)F_b(k-2;m,p) \\ &- mpF_b(k-3;m-1,p)\Big\} \end{aligned}$$

$$\begin{aligned} A_2 = \frac{1}{2}b_k^2\big((k-1)(k-2)&F_b(k-3;m,p) \\ &- 2(k-2)mpF_b(k-4;m-1,p) \\ &+ m(m-1)p^2F_b(k-5;m-2,p)\big) \end{aligned}$$

$$A_3 = \sum_{r=1}^{k-1} b_{2k-r}F_b^2(r-1;m,p)$$

$$\begin{aligned} A_4 = \sum_{r=2}^{k-1} b_{2k-r}b_r\big((r-1)&F_b(r-2;m,p) \\ &- mpF_b(r-3;m-1,p)\big) \end{aligned}$$

Given this approximation, we can calculate the probability that a packet is produced in a PGA of length N timesteps. From here, we can use a method such as interval bisection to calculate the length of PGA required to exceed the desired value of $p_{\texttt{packet}}$.

One implication of using this method for calculating the length of PGAs is the scaling with regards to different parameters. In particular, if we tighten the window or increase the value of $p_{\texttt{packet}}$, then the required length of the PGA grows rapidly, much faster than the value of $R_{\texttt{attempt}}$ decreases.

2) Calculating Minimum Rate using Hoeffding's inequality.

Let packet generation attempts occur at rate R, and succeed with probability p. Let the session have an acceptable probability of failure of at most $\epsilon_{service} \ll 1$, an expiry time of $t_{\texttt{expiry}}$ and require $N_{\texttt{inst}}$ instances to be successfully executed to obtain minimal service. Let $N = R \times t_{\texttt{expiry}}$, and let $N_{\texttt{inst}} = \alpha N$. Let S_N be the number of successfully executed instances.

Let $(X_i)_{i=1}^N \overset{iid}{\sim} \text{Bernoulli}(p)$. Then $S_N \overset{d}{=} \sum_{i=1}^N X_i$. Let M be the event that minimal service is obtained. Then

$$\begin{aligned} \mathbb{P}[M'] &= \mathbb{P}[S_N < N_{\texttt{inst}}] \\ &= \mathbb{P}[\mathbb{E}[S_N] - S_N > \mathbb{E}[S_N] - N_{\texttt{inst}}] \\ &= \mathbb{P}[pN - S_N > N(p-\alpha)] \\ &\leq \exp(-2N(p-\alpha)^2) \end{aligned} \tag{14}$$

where the last inequality is by Hoeffding's inequality [97].

The minimum number of packet generation attempts required is then given by

$$\min\left\{N \mid \epsilon_{service} > \exp(-2N(p-\alpha)^2) \wedge N > N_{\texttt{inst}}\right\} \tag{15}$$

from which we can then recover the minimum rate.

D. ADDITIONAL EVALUATION DETAILS

1) Entanglement Generation Model

Werner States We assume that all generated links are Werner states [98], and can be written in the format

$$\rho = \frac{1-F}{3}\mathbb{I}_2 + \frac{4F-1}{3}|\psi\rangle\langle\psi|. \tag{16}$$

where

$$|\psi\rangle = \frac{|00\rangle + |11\rangle}{2}$$

and F, the fidelity of ρ, is fixed.

The precise values we use are given in Table 2.

2) Entanglement Generation

In our implementation, we assume that there are network-wide timeslots, at the end of which an entangled link between the outer nodes and the central junction node is generated with known probability at a given fidelity. These are then instantaneously and deterministically swapped, if possible, to create the desired end-to-end entangled links. We also assume that the central node has no memories, and so it cannot store links for longer than one time-slot.

This means that there is only one possible scheme for generating end-to-end links, which is used by all pairs of nodes.

We also assume that all links are homogeneous, in particular that they all produce links of the same average fidelity.

3) Choice of cap on number of PGAs per schedule

To be able to estimate a suitable choice of cap on the number of PGAs per schedule, we computed 1125 network schedules. This was achieved by running the same simulation used for the evaluation, without a cap on the number of PGAs per schedule, and recording the number of PGAs which were scheduled and the time each schedule took to compute. We used the QKD application with requested packet rates of $R \in \{1.0, 1.1, 1.5\}$ and renewal rates $\lambda \in \{0.001, 0.0015, 0.002\}$. Each simulation lasted 7800 simulated seconds, for a total of 25 schedules computed in each simulation. We repeated each simulation 5 times.

Our implementation of the scheduling algorithm has complexity $O(N^2)$ where N is the number of PGAs to be scheduled. We therefore fitted a quadratic curve to the data and used this to obtain an estimate of the maximum number of PGAs which could be scheduled in under 150s (half the scheduling period). This can be seen in Figure 10.

From this we obtained an estimate of 1397, which we then rounded up to 1500 for neatness. This was done in part as we still had some uncertainty about the performance of the server whilst performing these simulations. In particular on other hardware we observed lower computational times, in which case the cap would be greater. Furthermore, in the `MDA` simulations, each PGT typically contributes between 150 and 300 PGAs to a schedule, and so increasing the cap by less than 150 will not allow more demands to be accepted from the queue.

4) Choice of Expiry times and Rates

Using the method in C2, we find that to have $\epsilon_{service} = 10^{-5}$, 850 PGAs should be scheduled. Therefore, we choose the rates to simulate such that at least 850 PGAs can be scheduled within $t_{\texttt{max duration}} - T_{SI}$ where T_{SI} is the scheduling interval.

In particular, we choose to simulate the end nodes requesting:

- An adaptive rate ($R = 0$).
- Fixed rates of:
  - Approximately the minimum rate required for 850 PGAs to be scheduled in $t_{\texttt{max duration}}$, $R^{min}_{\epsilon_{service}}$.
  - approximately $2R^{min}_{\epsilon_{service}}$

We choose to look at demands requesting a fixed packet generation rate of $R^{min}_{\epsilon_{service}}$, as it serves as a good comparison to the adaptive rate (which would use this rate in the absence of queuing). The choice of $2R^{min}_{\epsilon_{service}}$ gives us a motivated choice of a higher rate which we can compare against to see the impact of the choice of R on the performance metrics.

To choose the values of $t_{\texttt{max duration}}$, we look to the utilisation of the resulting PGTs when requesting the lowest fixed rate, $R^{min}_{\epsilon_{service}}$. As there are only 6 nodes in the network, if we choose a value of $t_{\texttt{max duration}}$ sufficiently long that the minimum rate gives a utilisation less than $\hat{\mathcal{U}}/5$, then all demands will always be immediately accepted and everyone will get minimal service almost surely. Likewise, if we set the value of $t_{\texttt{max duration}}$ too short, then only a couple of demands will be serviceable at a time without violating either SAC1 or SAC2, due to the high utilisation required. In a given deployment, we would expect that this is not desirable behaviour, and so such demands would be predominantly filtered out by the demand registration. Therefore, we choose the values of $t_{\texttt{max duration}}$ such that the utilisation of tasks requesting the slowest fixed rate is approximately 0.2, as then almost all demands can be satisfied simultaneously whilst the demand queue can still exert some influence over the performance of the network. We therefore choose $t^{\texttt{MDA}}_{\texttt{max duration}} = 2100s$ and $t^{\texttt{CKA}}_{\texttt{max duration}} = 4$ days. Note that with a scheduling interval $T^{\texttt{MDA}}_{SI} = 300$s, the choice of $t^{\texttt{MDA}}_{\texttt{max duration}} = 2100s$ results in an effective schedulable time of half an hour. For both `MDA` and `CKA`, we see the same trends for different values of $t_{\texttt{max duration}}$ whilst the same pressures from $\hat{\mathcal{U}}$ and the PGA cap exist.

5) Scheduler Model

We use the network scheduler as described in V-G, with a scheduling interval of 300 seconds for the `MDA` simulations and 3600 seconds (1 hour) for the `CKA` simulations. We take a longer scheduling interval for the `CKA` simulations as these demands last for much longer to serve than the `MDA` demands. Subsequently, this means that sessions are renewed on a much longer timescale and so the network state (schedule and demand queue) changes on a longer time scale.

6) Qoala files

Measure Directly Application (`MDA`)

```
META_START
    name: alice
    parameters: bob_id
    csockets: 0 -> bob
    epr_sockets: 0 -> bob
META_END

^b1 {type = QC}:
    tuple<m0> = run_request(tuple<>) : req

^b2 {type = CL}:
    return_result(m0)


REQUEST req
  callback_type: wait_all
  callback:
  return_vars: m0
  remote_id: {bob_id}
  epr_socket_id: 0
  num_pairs: 1
  virt_ids: all 0
  timeout: 1000
```

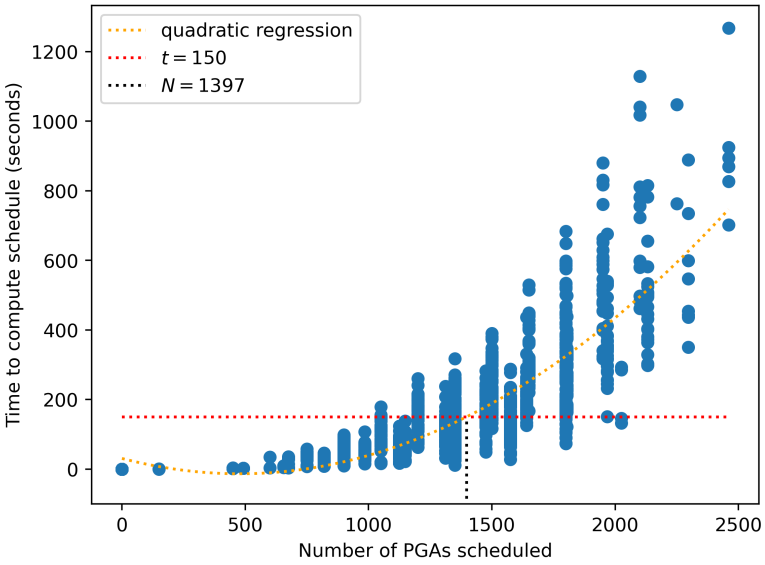


FIGURE 10: Times taken for our computational server to compute network schedules. The server has an Intel® Xeon® Gold 6230 CPU, with each core running at a maximum clock speed of 3.9GHz, and 189GB of random access memory. Each simulation was run on a single core, with up to 40 simulations being run in parallel. The regression modelling was performed using the `numpy` Python module

```
  fidelity: 1.0
  typ: measure_directly
  role: create
```

Listing 1: Qoala file for Alice

```
META_START
    name: alice
    parameters: bob_id
    csockets: 0 -> bob
    epr_sockets: 0 -> bob
META_END

^b1 {type = QC}:
    tuple<m0> = run_request(tuple<>) : req

^b2 {type = CL}:
    return_result(m0)


REQUEST req
  callback_type: wait_all
  callback:
  return_vars: m0
  remote_id: {bob_id}
  epr_socket_id: 0
  num_pairs: 1
  virt_ids: all 0
  timeout: 1000
  fidelity: 1.0
  typ: measure_directly
  role: create
```

Listing 2: Qoala file for Bob

Create and Keep Application (`CKA`)

```
META_START
    name: client
    parameters: server_id, alpha, beta, theta1,
        theta2
    csockets: 0 -> server
    epr_sockets: 0 -> server
META_END

^b0 {type = CL}:
    csocket = assign_cval() : 0

^b1 {type = QC}:
    run_request(tuple<>) : req0
^b2 {type = QL}:
    tuple<p2> = run_subroutine(tuple<theta2>) :
        post_epr_0
^b3 {type = QL}:
    tuple<p1> = run_subroutine(tuple<theta1>) :
        post_epr_1

^b4 {type = CL}:
    x = mult_const(p1) : 16
    minus_theta1 = mult_const(theta1) : -1
    delta1 = add_cval_c(minus_theta1, x)
    delta1 = add_cval_c(delta1, alpha)
    send_cmsg(csocket, delta1)

^b5 {type = CC}:
    m1 = recv_cmsg(csocket)

^b6 {type = CL}:
    y = mult_const(p2) : 16
    minus_theta2 = mult_const(theta2) : -1
    new_beta = bcond_mult_const(beta, m1) : -1
    delta2 = add_cval_c(new_beta, minus_theta2)
    delta2 = add_cval_c(delta2, y)
    send_cmsg(csocket, delta2)

    return_result(p1)
    return_result(p2)
```

```
SUBROUTINE post_epr_0
    params: theta2
    returns: p2
    uses: 0
    keeps:
    request:
  NETQASM_START
    load C0 @input[0]
    set Q0 0
    rot_z Q0 C0 4
    h Q0
    meas Q0 M0
    store M0 @output[0]
  NETQASM_END

SUBROUTINE post_epr_1
    params: theta1
    returns: p1
    uses: 1
    keeps:
    request:
  NETQASM_START
    load C0 @input[0]
    set Q0 1
    rot_z Q0 C0 4
    h Q0
    meas Q0 M1
    store M1 @output[0]
  NETQASM_END

REQUEST req0
  callback_type: wait_all
  callback:
  return_vars:
  remote_id: {server_id}
  epr_socket_id: 0
  num_pairs: 2
  window: 5_000_000
  virt_ids: increment 0
  timeout: 1000
  fidelity: 1.0
  typ: create_keep
  role: create
```

Listing 3: Qoala file for Client

```
META_START
    name: server
    parameters: client_id
    csockets: 0 -> client
    epr_sockets: 0 -> client
META_END

^b0 {type = CL}:
    csocket = assign_cval() : 0
    iterations = assign_cval() : 0

^b1 {type = QC}:
    run_request(tuple<>) : req0
^b2 {type = QL}:
    run_subroutine(tuple<>) : local_cphase

^b3 {type = CC}:
    delta1 = recv_cmsg(csocket)

^b4 {type = QL}:
    tuple<m1> = run_subroutine(tuple<delta1>) :
        meas_qubit_1

^b5 {type = CL}:
    send_cmsg(csocket, m1)

^b6 {type = CC}:
    delta2 = recv_cmsg(csocket)

^b7 {type = QL}:
    tuple<m2> = run_subroutine(tuple<delta2>) :
        meas_qubit_0

^b8 {type = CL}:
    return_result(m1)
    return_result(m2)



SUBROUTINE local_cphase
    params:
    returns:
    uses: 0, 1
    keeps: 0, 1
    request:
  NETQASM_START
    set Q0 1
    set Q1 0
    cphase Q0 Q1
  NETQASM_END

SUBROUTINE meas_qubit_1
    params: delta1
    returns: m1
    uses: 0, 1
    keeps: 0
    request:
  NETQASM_START
    load C0 @input[0]
    set Q1 1
    rot_z Q1 C0 4
    h Q1
    meas Q1 M0
    store M0 @output[0]
  NETQASM_END

SUBROUTINE meas_qubit_0
    params: delta2
    returns: m2
    uses: 0, 1
    keeps:
    request:
  NETQASM_START
    load C0 @input[0]
    set Q0 0
    rot_z Q0 C0 4
    h Q0
    meas Q0 M0
    store M0 @output[0]
  NETQASM_END

REQUEST req0
  callback_type: wait_all
  callback:
  return_vars:
  remote_id: {client_id}
  epr_socket_id: 0
  num_pairs: 2
  window: 5_000_000
  virt_ids: increment 0
  timeout: 1000
  fidelity: 1.0
  typ: create_keep
  role: receive
```

Listing 4: Qoala file for Server

7) Sensitivity to parameters

In order to speed up our simulations when testing the sensitivity of our simulations to the chosen parameters, we do not explicitly calculate the network schedule each time. To determine when a session obtains minimal service, it is simply required to know how many PGAs for a particular demand were scheduled in a given network schedule/scheduling interval. From this we can obtain the number of packets which were actually generated by sampling a $\mathrm{Binomial}(N, p_{\mathtt{packet}})$ distribution, and thereby establish whether the session obtained minimal service in that scheduling interval.

As our network scheduler schedules precisely one PGA per period of the PGT, given which demands/PGTs have been admitted by the scheduler admission control, we can calculate how many PGAs will be scheduled without having to calculate the schedule directly. Then as described above we can establish how many packets were generated and thus whether a session obtained minimal service.

We use this method of simulating the network scheduling for the additional data gathered for sensitivity testing of the simulation parameters. We also validated it against computing the network schedule and saw a perfect match for the same datasets.

8) Calculating the expected queuing time

We use the notation from § VI-D. Let $t_{renew} \sim \mathrm{Exponential}(\lambda)$. Then the time which a demand $\mathcal{D}$ would sit in the demand queue for in the absence of other demands is $Q \equiv T_{SI} - t_{renew} \mod T_{SI}$. For convenience we write $T_{SI} = T$, and let $t_{\mathcal{D}}$ be the time demand $\mathcal{D}$ is submitted. We then calculate:

$$\begin{aligned} F_Q(\tau) &= \mathbb{P}[Q \leq \tau] \\ &= \mathbb{P}\left[t_{\mathcal{D}} \in \bigcup_{k=1}^{\infty} [kT - \tau, kT]\right] \\ &= \sum_{k=1}^{\infty} \int_{kT-\tau}^{kT} \lambda e^{-\lambda x} \mathrm{d}x \\ &= \sum_{k=1}^{\infty} e^{-kT\lambda}(-1 + e^{\lambda\tau}) \\ &= \frac{-1 + e^{\lambda\tau}}{-1 + e^{\lambda T}} \end{aligned} \tag{17}$$

$$\begin{aligned} \mathbb{E}[Q] &= \int_0^T \tau \ \mathrm{d}F_Q \\ &= \int_0^T \frac{\tau\lambda e^{\lambda\tau}}{-1 + e^{\lambda T}} \ \mathrm{d}\tau \\ &= T\left(1 - \frac{1}{\lambda T} + \frac{1}{e^{\lambda T} - 1}\right) \end{aligned} \tag{18}$$

ACKNOWLEDGEMENTS

TB, HJ, SG and SW acknowledge funding from the Quantum Internet Alliance (QIA). QIA has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No. 101102140. SW also acknowledges funding from NWO VICI. We thank Wojciech Kozlowski and Ingmar te Raa for critical feedback on the content of this manuscript and thank Bart van der Vecht for advice about Qoala.

REFERENCES

[1] A. K. Ekert, "Quantum cryptography based on Bell's theorem," *Physical Review Letters*, vol. 67, no. 6, pp. 661–663, Aug. 1991. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.67.661

[2] C. H. Bennett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing," in *Proceedings of the International Conference on Computers, Systems & Signal Processing, Bangalore, India*, vol. 1. IEEE, 1984, pp. 175–179.

[3] P. Arright and L. Salvail, "Blind quantum computation," *International Journal of Quantum Information*, vol. 04, no. 05, pp. 883–898, 2006.

[4] A. Broadbent, J. Fitzsimons, and E. Kashefi, "Universal Blind Quantum Computation," in *2009 50th Annual IEEE Symposium on Foundations of Computer Science*. Atlanta, Georgia, USA: IEEE, Oct. 2009, pp. 517–526.

[5] A. M. Childs, "Secure assisted quantum computation," *Quantum Information and Computation*, vol. 5, no. 6, 2005, arXiv:quant-ph/0111046. [Online]. Available: http://arxiv.org/abs/quant-ph/0111046

[6] S. Tani, H. Kobayashi, and K. Matsumoto, "Exact Quantum Algorithms for the Leader Election Problem," *ACM Transactions on Computation Theory*, vol. 4, no. 1, pp. 1–24, Mar. 2012. [Online]. Available: https://dl.acm.org/doi/10.1145/2141938.2141939

[7] S. Wehner, D. Elkouss, and R. Hanson, "Quantum internet: A vision for the road ahead," *Science*, Oct. 2018, publisher: American Association for the Advancement of Science. [Online]. Available: https://www.science.org/doi/10.1126/science.aam9288

[8] A. Einstein, B. Podolsky, and N. Rosen, "Can quantum-mechanical description of physical reality be considered complete?" *Physical Review*, vol. 47, no. 10, pp. 777–780, 1935. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRev.47.777

[9] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010, ISBN: 9780511976667 Publisher: Cambridge University Press. [Online]. Available: https://www.cambridge.org/highereducation/books/quantum-computation-and-quantum-information/01E10196D0A682A6AEFFEA52D53BE9AE

[10] W. Dür, H.-J. Briegel, J. I. Cirac, and P. Zoller, "Quantum repeaters based on entanglement purification," *Physical Review A*, vol. 59, no. 1, pp. 169–181, Jan. 1999, publisher: American Physical Society. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.59.169

[11] M. Pompili, S. L. N. Hermans, S. Baier, H. K. C. Beukers, P. C. Humphreys, R. N. Schouten, R. F. L. Vermeulen, M. J. Tiggelman, L. dos Santos Martins, B. Dirkse, S. Wehner, and R. Hanson, "Realization of a multinode quantum network of remote solid-state qubits," *Science*, vol. 372, no. 6539, pp. 259–264, Apr. 2021, publisher: American Association for the Advancement of Science. [Online]. Available: https://www.science.org/doi/10.1126/science.abg1919

[12] Á. G. Iñesta, G. Vardoyan, L. Scavuzzo, and S. Wehner, "Optimal entanglement distribution policies in homogeneous repeater chains with cutoffs," *npj Quantum Information*, vol. 9, no. 1, pp. 1–7, May 2023, number: 1 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/s41534-023-00713-9

[13] C. Cabrillo, J. I. Cirac, P. García-Fernández, and P. Zoller, "Creation of entangled states of distant atoms by interference," *Physical Review A*, vol. 59, no. 2, pp. 1025–1033, Feb. 1999. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.59.1025

[14] C. E. Bradley, S. W. de Bone, P. F. W. Moller, S. Baier, M. J. Degen, S. J. H. Loenen, H. P. Bartling, M. Markham, D. J. Twitchen, R. Hanson, D. Elkouss, and T. H. Taminiau, "Robust quantum-network memory based on spin qubits in isotopically engineered diamond," Nov. 2021, arXiv:2111.09772 [cond-mat, physics:quant-ph]. [Online]. Available: http://arxiv.org/abs/2111.09772

[15] C. D. Donne, M. Iuliano, B. van der Vecht, G. M. Ferreira, H. Jirovská, T. van der Steenhoven, A. Dahlberg, M. Skrzypczyk, D. Fioretto, M. Teller, P. Filippov, A. R.-P. Montblanch, J. Fischer, B. van Ommen,

N. Demetriou, D. Leichtle, L. Music, H. Ollivier, I. te Raa, W. Kozlowski, T. Taminiau, P. Pawełczak, T. Northup, R. Hanson, and S. Wehner, "Design and demonstration of an operating system for executing applications on quantum network nodes," 2024. [Online]. Available: https://arxiv.org/abs/2407.18306

[16] B. van der Vecht, A. T. Yücel, H. Jirovská, and S. Wehner, "Qoala: an application execution environment for quantum internet nodes," 2025, arXiv:2502.17296 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2502.17296

[17] T. R. Beauchamp, H. Jirovská, S. Gauthier, and S. Wehner, "Data and code for "a modular quantum network architecture for integrating network scheduling with local program execution."," 2025, doi: 10.4121/71317dee-089c-4507-8f85-6a4718c4b8d7.

[18] T. R. Beauchamp, H. Jirovská, S. Gauthier, and S. Wehner, "Data for 'a modular quantum network architecture for integrating network scheduling with local program execution.'," 2025, doi: 10.4121/99NA-4P24.

[19] H. Gu, R. Yu, Z. Li, X. Wang, and F. Zhou, "Esdi: Entanglement scheduling and distribution in the quantum internet," *2023 32nd International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–10, 2023.

[20] A. Pirker and W. Dür, "A quantum network stack and protocols for reliable entanglement-based networks," *New Journal of Physics*, vol. 21, no. 3, p. 033003, 2019, publisher: IOP Publishing. [Online]. Available: https://dx.doi.org/10.1088/1367-2630/ab05f7

[21] C. Zhan, J. Chung, A. Zang, A. Kolar, and R. Kettimuthu, "Design and simulation of the adaptive continuous entanglement generation protocol," in *2025 International Conference on Quantum Communications, Networking, and Computing (QCNC)*, 2025, pp. 127–134.

[22] L. Talsma, Á. G. Iñesta, and S. Wehner, "Continuously distributing entanglement in quantum networks with regular topologies," *Physical Review A*, vol. 110, no. 2, Aug. 2024. [Online]. Available: http://dx.doi.org/10.1103/PhysRevA.110.022429

[23] M. Pompili, C. Delle Donne, I. te Raa, B. van der Vecht, M. Skrzypczyk, G. Ferreira, L. de Kluijver, A. J. Stolk, S. L. N. Hermans, P. Pawełczak, W. Kozlowski, R. Hanson, and S. Wehner, "Experimental demonstration of entanglement delivery using a quantum network stack," *npj Quantum Information*, vol. 8, no. 1, pp. 1–10, Oct. 2022, number: 1 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/s41534-022-00631-2

[24] A. Dahlberg, M. Skrzypczyk, T. Coopmans, L. Wubben, F. Rozpędek, M. Pompili, A. Stolk, P. Pawełczak, R. Knegjens, J. d. O. Filho, R. Hanson, and S. Wehner, "A Link Layer Protocol for Quantum Networks," in *Proceedings of the ACM Special Interest Group on Data Communication*, Aug. 2019, pp. 159–173, arXiv:1903.09778 [quant-ph]. [Online]. Available: http://arxiv.org/abs/1903.09778

[25] B. Davies, T. Beauchamp, G. Vardoyan, and S. Wehner, "Tools for the analysis of quantum protocols requiring state generation within a time window," *IEEE Transactions on Quantum Engineering*, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10417724/

[26] J. I. Naus, "Approximations for Distributions of Scan Statistics," *Journal of the American Statistical Association*, vol. 77, no. 377, pp. 177–183, 1982, publisher: [American Statistical Association, Taylor & Francis, Ltd.]. [Online]. Available: https://www.jstor.org/stable/2287786

[27] J. Glaz, J. I. Naus, and S. Wallenstein, *Scan statistics*, ser. Springer series in statistics. New York: Springer, 2001.

[28] M. Skrzypczyk and S. Wehner, "An Architecture for Meeting Quality-of-Service Requirements in Multi-User Quantum Networks," Nov. 2021, arXiv:2111.13124 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2111.13124

[29] C. Cicconetti, M. Conti, and A. Passarella, "Request Scheduling in Quantum Networks," *IEEE Transactions on Quantum Engineering*, vol. 2, pp. 2–17, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9461156/

[30] R. Van Meter, R. Satoh, N. Benchasattabuse, T. Matsuo, M. Hajdušek, T. Satoh, S. Nagayama, and S. Suzuki, "A Quantum Internet Architecture," in *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, Sep. 2022, pp. 341–352, arXiv:2112.07092 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2112.07092

[31] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015, conference Name: Proceedings of the IEEE.

[32] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: https://dl.acm.org/doi/10.1145/1355734.1355746

[33] J. M. Halpern, R. HAAS, a. doria, L. Dong, W. Wang, H. M. Khosravi, J. H. Salim, and R. Gopal, "Forwarding and Control Element Separation (ForCES) Protocol Specification," Internet Engineering Task Force, Request for Comments RFC 5810, Mar. 2010, num Pages: 124. [Online]. Available: https://datatracker.ietf.org/doc/rfc5810

[34] W. Kozlowski, F. Kuipers, and S. Wehner, "A p4 data plane for the quantum internet," in *Proceedings of the 3rd P4 Workshop in Europe*. ACM, 2020, pp. 49–51. [Online]. Available: https://dl.acm.org/doi/10.1145/3426744.3431321

[35] Z. Yang and C. Cui, "Reconfigurable Quantum Internet Service Provider," May 2023, arXiv:2305.09048 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2305.09048

[36] J. Chung, E. M. Eastman, G. S. Kanter, K. Kapoor, N. Lauk, C. H. Peña, R. K. Plunkett, N. Sinclair, J. M. Thomas, R. Valivarthi, S. Xie, R. Kettimuthu, P. Kumar, P. Spentzouris, and M. Spiropulu, "Design and implementation of the illinois express quantum metropolitan area network," *IEEE Transactions on Quantum Engineering*, vol. 3, pp. 1–20, 2022.

[37] R. S. Tessinari, R. I. Woodward, and A. J. Shields, "Software-defined quantum network using a QKD-secured SDN controller and encrypted messages," May 2023, arXiv:2305.12893 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2305.12893

[38] V. Martin, J. P. Brito, L. Ortiz, R. B. Mendez, J. S. Buruaga, R. J. Vicente, A. Sebastián-Lombraña, D. Rincon, F. Perez, C. Sanchez, M. Peev, H. H. Brunner, F. Fung, A. Poppe, F. Fröwis, A. J. Shields, R. I. Woodward, H. Griesser, S. Roehrich, F. De La Iglesia, C. Abellan, M. Hentschel, J. M. Rivas-Moscoso, A. Pastor, J. Folgueira, and D. R. Lopez, "MadQCI: a heterogeneous and scalable SDN QKD network deployed in production facilities." [Online]. Available: http://arxiv.org/abs/2311.12791

[39] A. Aguado, V. Martin, D. Lopez, M. Peev, J. Martinez-Mateo, J. L. Rosales, F. de la Iglesia, M. Gomez, E. Hugues Salas, A. Lord, R. Nejabati, and D. Simeonidou, "Quantum-aware software defined networks," in *6th International Conference on Quantum Cryptography (QCRYPT 2016)*. QCrypt, 2016.

[40] W. Yu, B. Zhao, and Z. Yan, "Software defined quantum key distribution network," in *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*. IEEE, 2017, pp. 1293–1297. [Online]. Available: http://ieeexplore.ieee.org/document/8322751/

[41] A. Aguado, E. Hugues-Salas, P. A. Haigh, J. Marhuenda, A. B. Price, P. Sibson, J. E. Kennard, C. Erven, J. G. Rarity, M. G. Thompson, A. Lord, R. Nejabati, and D. Simeonidou, "Secure NFV orchestration over an SDN-controlled optical network with time-shared quantum key distribution resources," *Journal of Lightwave Technology*, vol. 35, no. 8, pp. 1357–1362, 2017.

[42] H. Wang, Y. Zhao, and A. Nag, "Quantum-key-distribution (QKD) networks enabled by software-defined networks (SDN)," *Applied Sciences*, vol. 9, no. 10, p. 2081, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/10/2081

[43] X. Wang, T. Chen, Y. Du, J. He, and C. Zhu, "Network slicing-based and QoS-oriented software-defined quantum key distribution network," in *2023 International Conference on Ubiquitous Communication (Ucom)*, 2023, pp. 397–402. [Online]. Available: https://ieeexplore.ieee.org/document/10257658

[44] H. S. Hadi and A. J. Obaid, "Quantum key distribution (QKD) for wireless networks with software-defined networking," *Internet Technology Letters*, vol. n/a, p. e547, 2024, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/itl2.547. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/itl2.547

[45] M. Iqbal, A. Villegas, M. S. Moreolo, L. Nadal, R. Muñoz, P. Adillon, S. Sarmiento, J. Tabares, and S. Etcheverry, "SDN-enabled continuous-variable QKD in coexistence with 8×200 gb/s 16-QAM classical channels," in *2024 International Conference on Optical Network Design and Modeling (ONDM)*, 2024, pp. 1–3. [Online]. Available: https://ieeexplore.ieee.org/document/10582669/authors#authors

[46] S. Gupta, I. Agarwal, V. Mogiligidda, R. Kumar Krishnan, S. Chennuri, D. Aggarwal, A. Hoodati, S. Cooper, Ranjan, M. Bilal Sheik, K. M. Bhavya, M. Hegde, M. N. Krishna, A. K. Chauhan, M. Korrapati, S. Singh, J. B. Singh, S. Sud, S. Gupta, S. Pant, Sankar, N. Agrawal, A. Ranjan, P. Mohapatra, T. Roopak, A. Ahmad, M. Nanjunda, and D. Singh, "ChaQra: a cellular unit of the indian quantum network," *Scientific Reports*, vol. 14, no. 1, p. 16752, 2024, publisher: Nature

Publishing Group. [Online]. Available: https://www.nature.com/articles/s41598-024-67495-8

[47] Quantum Key Distribution Industry Specification Group, "Quantum key distribution (QKD): Control interface for software defined networks," European Telecommunications Standards Institute, Tech. Rep., 2024. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/QKD/001_099/015/02.01.01_60/gs_QKD015v020101p.pdf

[48] S. S. Craciunas and R. S. Oliver, "Combined task- and network-level scheduling for distributed time-triggered systems," *Real-Time Systems*, vol. 52, no. 2, pp. 161–200, 2016. [Online]. Available: http://link.springer.com/10.1007/s11241-015-9244-x

[49] H. J. Kramer, "TTEthernet (time-triggered ethernet) - eoPortal," retrieved from https://www.eoportal.org/satellite-missions/ttethernet on 2023-10-30. [Online]. Available: https://www.eoportal.org/satellite-missions/ttethernet

[50] TTTech, "NASA's orion spacecraft," retrieved from: https://www.tttech.com/aerospace/resources/case-studies/nasa-orion on 2023-10-30. [Online]. Available: https://www.tttech.com/aerospace/resources/case-studies/nasa-orion

[51] M. Ruf, N. H. Wan, H. Choi, D. Englund, and R. Hanson, "Quantum networks based on color centers in diamond," *Journal of Applied Physics*, vol. 130, no. 7, p. 070901, Aug. 2021. [Online]. Available: https://doi.org/10.1063/5.0056534

[52] M. Pompili, S. L. N. Hermans, S. Baier, H. K. C. Beukers, P. C. Humphreys, R. N. Schouten, R. F. L. Vermeulen, M. J. Tiggelman, L. dos Santos Martins, B. Dirkse, S. Wehner, and R. Hanson, "Realization of a multinode quantum network of remote solid-state qubits," *Science*, vol. 372, no. 6539, pp. 259–264, Apr. 2021, publisher: American Association for the Advancement of Science. [Online]. Available: https://www.science.org/doi/10.1126/science.abg1919

[53] V. Krutyanskiy, M. Galli, V. Krcmarsky, S. Baier, D. A. Fioretto, Y. Pu, A. Mazloom, P. Sekatski, M. Canteri, M. Teller, J. Schupp, J. Bate, M. Meraner, N. Sangouard, B. P. Lanyon, and T. E. Northup, "Entanglement of trapped-ion qubits separated by 230 meters," *Phys. Rev. Lett.*, vol. 130, p. 050803, Feb. 2023. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.130.050803

[54] P. Maunz, D. L. Moehring, S. Olmschenk, K. C. Younge, D. N. Matsukevich, and C. Monroe, "Quantum interference of photon pairs from two remote trapped atomic ions," *Nature Physics*, vol. 3, no. 8, pp. 538–541, 2007.

[55] J. P. Covey, H. Weinfurter, and H. Bernien, "Quantum networks with neutral atom processing nodes," *npj Quantum Information*, vol. 9, no. 1, pp. 1–12, Sep. 2023, number: 1 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/s41534-023-00759-9

[56] M. Uphoff, M. Brekenfeld, G. Rempe, and S. Ritter, "An integrated quantum repeater at telecom wavelength with single atoms in optical fiber cavities," *Applied Physics B*, vol. 122, no. 3, Mar. 2016. [Online]. Available: http://dx.doi.org/10.1007/s00340-015-6299-2

[57] C. W. Chou, H. de Riedmatten, D. Felinto, S. V. Polyakov, S. J. van Enk, and H. J. Kimble, "Measurement-induced entanglement for excitation stored in remote atomic ensembles," *Nature*, vol. 438, no. 7069, pp. 828–832, Dec. 2005.

[58] C. W. Chou, J. Laurat, H. Deng, K. S. Choi, H. de Riedmatten, D. Felinto, and H. J. Kimble, "Functional quantum nodes for entanglement distribution over scalable quantum networks," *Science*, vol. 316, no. 5829, pp. 1316–1320, Jun. 2007. [Online]. Available: https://doi.org/10.1126%2Fscience.1140300

[59] P. Drmota, D. Nadlinger, D. Main, B. Nichol, E. Ainley, D. Leichtle, A. Mantri, E. Kashefi, R. Srinivas, G. Araneda, C. Ballance, and D. Lucas, "Verifiable blind quantum computing with trapped ions and single photons," *Physical Review Letters*, vol. 132, no. 15, Apr. 2024. [Online]. Available: http://dx.doi.org/10.1103/PhysRevLett.132.150604

[60] R. C. Berrevoets, T. Middelburg, R. F. L. Vermeulen, L. D. Chiesa, F. Broggi, S. Piciaccia, R. Pluis, P. Umesh, J. F. Marques, W. Tittel, and J. A. Slater, "Deployed measurement-device independent quantum key distribution and bell-state measurements coexisting with standard internet data and networking equipment," *Communications Physics*, vol. 5, no. 1, p. 186, 2022. [Online]. Available: https://doi.org/10.1038/s42005-022-00964-6

[61] G. Vardoyan, S. Guha, P. Nain, and D. Towsley, "On the Stochastic Analysis of a Quantum Entanglement Distribution Switch," *IEEE Transactions on Quantum Engineering*, vol. 2, pp. 1–16, 2021.

[62] ——, "On the exact analysis of an idealized quantum switch," *Performance Evaluation*, vol. 144, p. 102141, 2020.

[63] N. K. Panigrahy, T. Vasantam, D. Towsley, and L. Tassiulas, "On the capacity region of a quantum switch with entanglement purification," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 2023, pp. 1–10.

[64] Y. Lee, E. Bersin, A. Dahlberg, S. Wehner, and D. Englund, "A quantum router architecture for high-fidelity entanglement flows in quantum networks," *npj Quantum Information*, vol. 8, no. 1, Jun. 2022. [Online]. Available: http://dx.doi.org/10.1038/s41534-022-00582-8

[65] S. Gauthier, G. Vardoyan, and S. Wehner, "An architecture for control of entanglement generation switches in quantum networks," *IEEE Transactions on Quantum Engineering*, vol. 4, pp. 1–17, 2023.

[66] S. Gauthier, T. Vasantam, and G. Vardoyan, "An on-demand resource allocation algorithm for a quantum network hub and its performance analysis," 2024. [Online]. Available: https://arxiv.org/abs/2405.18066

[67] L.-M. Duan, M. D. Lukin, J. I. Cirac, and P. Zoller, "Long-distance quantum communication with atomic ensembles and linear optics," *Nature*, vol. 414, no. 6862, pp. 413–418, Nov. 2001.

[68] C. Simon, H. de Riedmatten, M. Afzelius, N. Sangouard, H. Zbinden, and N. Gisin, "Quantum Repeaters with Photon Pair Sources and Multimode Memories," *Phys. Rev. Lett.*, vol. 98, p. 190503, May 2007.

[69] N. Sangouard, C. Simon, H. de Riedmatten, and N. Gisin, "Quantum repeaters based on atomic ensembles and linear optics," *Rev. Mod. Phys.*, vol. 83, pp. 33–80, Mar 2011.

[70] N. Sangouard, R. Dubessy, and C. Simon, "Quantum repeaters based on single trapped ions," *Phys. Rev. A*, vol. 79, p. 042340, Apr 2009.

[71] L. Kamin, E. Shchukin, F. Schmidt, and P. van Loock, "Exact rate analysis for quantum repeaters with imperfect memories and entanglement swapping as soon as possible." [Online]. Available: http://arxiv.org/abs/2203.10318

[72] H. Zimmermann, "Osi reference model - the iso model of architecture for open systems interconnection," *IEEE Transactions on Communications*, vol. 28, no. 4, pp. 425–432, 1980.

[73] C. D. Donne, M. Iuliano, B. van der Vecht, G. M. Ferreira, H. Jirovská, T. van der Steenhoven, A. Dahlberg, M. Skrzypczyk, D. Fioretto, M. Teller, P. Filippov, A. R.-P. Montblanch, J. Fischer, B. van Ommen, N. Demetriou, D. Leichtle, L. Music, H. Ollivier, I. t. Raa, W. Kozlowski, T. Taminiau, P. Pawełczak, T. Northup, R. Hanson, and S. Wehner, "Design and demonstration of an operating system for executing applications on quantum network nodes," Jul. 2024, arXiv:2407.18306 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2407.18306

[74] H. Bernien, B. Hensen, W. Pfaff, G. Koolstra, M. S. Blok, L. Robledo, T. H. Taminiau, M. Markham, D. J. Twitchen, L. Childress, and R. Hanson, "Heralded entanglement between solid-state qubits separated by three metres," *Nature*, vol. 497, no. 7447, pp. 86–90, Apr. 2013.

[75] P. C. Humphreys, N. Kalb, J. P. J. Morits, R. N. Schouten, R. F. L. Vermeulen, D. J. Twitchen, M. Markham, and R. Hanson, "Deterministic delivery of remote entanglement on a quantum network," *Nature*, vol. 558, no. 7709, pp. 268–273, Jun. 2018, arXiv:1712.07567 [quant-ph]. [Online]. Available: http://arxiv.org/abs/1712.07567

[76] T. van Leent, M. Bock, F. Fertig, R. Garthoff, S. Eppelt, Y. Zhou, P. Malik, M. Seubert, T. Bauer, W. Rosenfeld, W. Zhang, C. Becher, and H. Weinfurter, "Entangling single atoms over 33 km telecom fibre," *Nature*, vol. 607, no. 7917, pp. 69–73, Jul. 2022.

[77] N. Joukov, A. Traeger, R. Iyer, C. P. Wright, and E. Zadok, "Operating system profiling via latency analysis," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, ser. OSDI '06. USA: USENIX Association, 2006, p. 89–102.

[78] A. S. Tanenbaum, *Modern Operating Systems*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[79] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 3rd ed. New York, NY, USA: Springer Science + Business Media, 2011.

[80] S. Shenker, "Fundamental design issues for the future Internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 7, pp. 1176–1188, Sep. 1995, conference Name: IEEE Journal on Selected Areas in Communications.

[81] A. Furusawa, J. L. Sørensen, S. L. Braunstein, C. A. Fuchs, H. J. Kimble, and E. S. Polzik, "Unconditional quantum teleportation," *Science*, vol. 282, no. 5389, pp. 706–709, 1998. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.282.5389.706

[82] M. D. Barrett, J. Chiaverini, T. Schaetz, J. Britton, W. M. Itano, J. D. Jost, E. Knill, C. Langer, D. Leibfried, R. Ozeri, and D. J. Wineland,

"Deterministic quantum teleportation of atomic qubits," *Nature*, vol. 429, pp. 737–739, 2004.

[83] D. Leichtle, L. Music, E. Kashefi, and H. Ollivier, "Verifying BQP Computations on Noisy Devices with Minimal Overhead," Sep. 2021, arXiv:2109.04042 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2109.04042

[84] F. B. Talbot, "Resource-Constrained Project Scheduling with Time-Resource Tradeoffs: The Nonpreemptive Case," *Management Science*, vol. 28, no. 10, pp. 1197–1210, 1982, publisher: INFORMS. [Online]. Available: https://www.jstor.org/stable/2630948

[85] H. Jirovská, "Evaluating an RCPSP Implementation of Quantum Program Scheduling," Master's thesis, Technische Universiteit Delft, Delft, Jun. 2023. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid%3A6abe7dc4-0308-490e-b156-eed07426e129

[86] A. Green, J. Lawrence, G. Siopsis, N. A. Peters, and A. Passian, "Quantum key distribution for critical infrastructures: Towards cyber-physical security for hydropower and dams," *Sensors*, vol. 23, no. 24, p. 9818, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/24/9818

[87] M. Alshowkan, P. G. Evans, M. Starke, D. Earl, and N. A. Peters, "Authentication of smart grid communications using quantum key distribution," *Scientific Reports*, vol. 12, no. 1, p. 12731, 2022. [Online]. Available: https://www.nature.com/articles/s41598-022-16090-w

[88] S. Aggarwal and G. Kaddoum, "Authentication of smart grid by integrating QKD and blockchain in SCADA systems," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10591785/

[89] M. Pant, H. Krovi, D. Towsley, L. Tassiulas, L. Jiang, P. Basu, D. Englund, and S. Guha, "Routing entanglement in the quantum internet," *npj Quantum Information*, vol. 5, no. 1, p. 25, Dec. 2019. [Online]. Available: http://www.nature.com/articles/s41534-019-0139-x

[90] R. Van Meter, T. Satoh, T. D. Ladd, W. J. Munro, and K. Nemoto, "Path Selection for Quantum Repeater Networks," *Networking Science*, vol. 3, no. 1-4, pp. 82–95, Dec. 2013, arXiv:1206.5655 [quant-ph]. [Online]. Available: http://arxiv.org/abs/1206.5655

[91] J. A. Stankovic, M. Spuri, G. C. Buttazzo, and K. Ramamritham, *Deadline scheduling for real-time systems: EDF and related algorithms*, ser. The Kluwer international series in engineering and computer science. New York: Kluwer Academic Publishers, Dec. 1999, no. SECS 460. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S089812219991235X

[92] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973. [Online]. Available: https://dl.acm.org/doi/10.1145/321738.321743

[93] S. Low and D. Lapsley, "Optimization flow control. i. basic algorithm and convergence," *IEEE/ACM Transactions on Networking*, vol. 7, no. 6, pp. 861–874, 1999.

[94] K. Steenhaut, K. Degieter, W. Brissinck, and E. Dirkx, "Scheduling and admission control policies: A case study for ATM," *Computer Networks and ISDN Systems*, vol. 29, no. 5, pp. 539–554, 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169755296001183

[95] M. Christandl and S. Wehner, "Quantum Anonymous Transmissions," in *Advances in Cryptology - ASIACRYPT 2005*, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, and B. Roy, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, vol. 3788, pp. 217–235, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/11593447_12

[96] V. Lipinska, G. Murta, and S. Wehner, "Anonymous transmission in a noisy quantum network using the W state," *Physical Review A*, vol. 98, no. 5, p. 052320, Nov. 2018. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.98.052320

[97] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 13–30, 1963. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/01621459.1963.10500830

[98] R. F. Werner, "Quantum states with Einstein-Podolsky-Rosen correlations admitting a hidden variable model," *Physical Review A*, vol. 40, no. 8, pp. 4277–4281, Oct. 1989. [Online]. Available: http://www2.fisica.unlp.edu.ar/materias/qc/Sp.pdf