TU Delft

Delft University of Technology

Document Version
Final published version

Licence
CC BY-NC-ND

Citation (APA)
Gauthier, S. S. (2026). *Control Architectures for Multi-User Quantum Networks*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:189088f3-707d-4d1e-a6c4-c71b817369e7

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

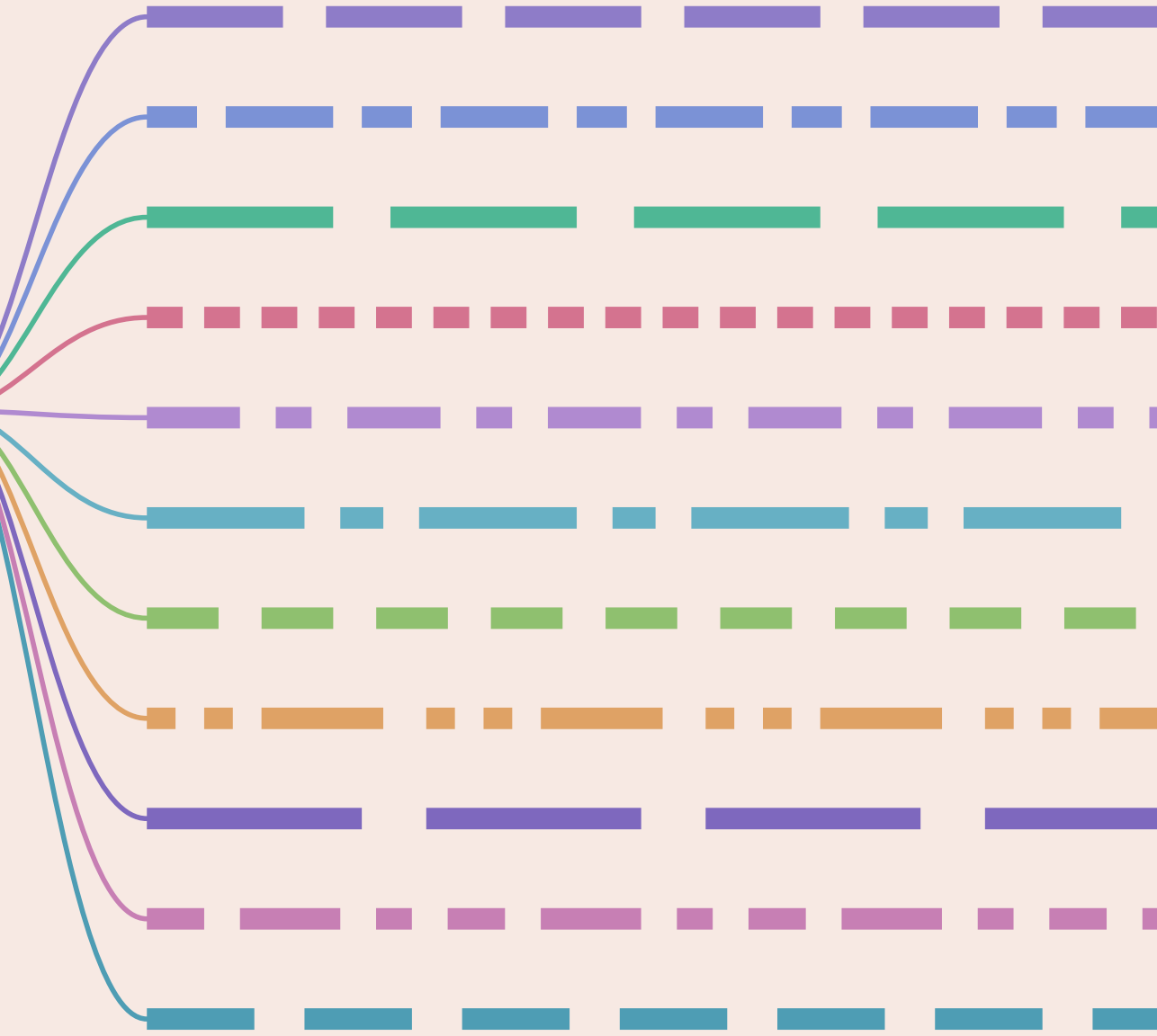
Copyright
In case the licence states “Dutch Copyright Act (Article 25fa)”, this publication was made available Green Open Access via the TU Delft Institutional Repository pursuant to Dutch Copyright Act (Article 25fa, the Taverne amendment). This provision does not affect copyright ownership.
Unless copyright is transferred by contract or statute, it remains with the copyright holder.

Sharing and reuse
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.

Control Architectures for Multi-User Quantum Networks



Scarlett Savannah Gauthier

CONTROL ARCHITECTURES FOR MULTI-USER QUANTUM NETWORKS

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology,
by the authority of the Rector Magnificus prof. dr. ir. H. Bijl,
chair of the Board for Doctorates,
to be defended publicly Monday 14 September 2026 at 12:30 pm.

by

Scarlett Savannah GAUTHIER

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof. dr. S.D.C. Wehner,	Delft University of Technology, promotor
Prof. dr. ir. R. Hanson,	Delft University of Technology, promotor
Independent members:	
Prof. dr. Y. M. Blanter,	Delft University of Technology
Prof. dr. ir. F. A. Kuipers,	Delft University of Technology
Prof. dr. M. Amoretti,	University of Parma, Italy
Dr. R. Hai,	Delft University of Technology
Dr. A. Pappa,	Technical University of Berlin, Germany
Prof. dr. ir. L. M. K. Vandersypen,	Delft University of Technology, reserve member

Copyright © 2026 Scarlett Savannah Gauthier

ISBN (e-book PDF) 978-94-6518-400-5
ISBN (print) 978-94-6563-003-8

Printed by: Ridderprint | www.ridderprint.nl

An electronic version of this dissertation will be made available at
https://repository.tudelft.nl.

There is no reason to assume the existence of shortcuts.

John von Neumann and Oskar Morgenstern

CONTENTS

Summary 1

Samenvatting 3

List of Publications 5

List of Patents 7

1 Introduction 9
- 1.1 The Promise of Quantum Networks . . . 9
- 1.2 Quantum Information . . . 10
  - 1.2.1 Fundamentals and Representation . . . 10
  - 1.2.2 Correlations and Entanglement . . . 11
  - 1.2.3 No-Cloning . . . 12
- 1.3 Quantum Networks as Service Providers . . . 13
- 1.4 Entanglement Generation . . . 13
  - 1.4.1 Methods . . . 13
  - 1.4.2 Bridging Long Distances . . . 15
  - 1.4.3 Storage of Entangled Links . . . 15
- 1.5 Types of Quantum Network . . . 15
  - 1.5.1 Modalities . . . 15
  - 1.5.2 Application Programming Paradigms . . . 17
  - 1.5.3 A Classification of Entanglement-Based Networks . . . 18
- 1.6 Protocols and Architectures . . . 20
  - 1.6.1 Protocols . . . 20
  - 1.6.2 Architectures . . . 20
  - 1.6.3 Network Protocol Stack . . . 21
  - 1.6.4 Quantum Network Control Architectures . . . 22
- 1.7 Enabling Multi-User Access . . . 22
- 1.8 Scaling From Demonstrations to Deployments . . . 23
- 1.9 A Prototype Full-Stack Quantum Network . . . 24
- 1.10 Problem Statement . . . 25
- 1.11 Thesis Overview . . . 26

References . . . 27

2 An Architecture for Control of Entanglement Generation Switches in Quantum Networks 39
- 2.1 Introduction . . . 39
  - 2.1.1 Results Summary . . . 41
  - 2.1.2 Related Work . . . 41

2.2 Preliminaries . . . 44
2.2.1 Demands for resource allocation from nodes to the EGS processor . 45
2.2.2 Processing demands for resource allocation . . . 45
2.2.3 Constraints . . . 48
2.2.4 Mathematical Background . . . 48
2.3 RCP Algorithm . . . 49
2.3.1 Derivation . . . 51
2.3.2 Convergence . . . 54
2.4 Case Study . . . 54
2.5 Discussion . . . 57
2.6 Proofs . . . 59
References . . . 67

3 On-Demand Resource Allocation for a Quantum Network Hub 71
3.1 Introduction . . . 71
3.2 Background . . . 75
3.3 System Description . . . 77
3.3.1 Physical Operation Settings . . . 78
3.4 Model and Assumptions . . . 80
3.4.1 System Model and Notation . . . 81
3.4.2 Session Structure . . . 83
3.4.3 Service Models . . . 84
3.5 Analysis . . . 87
3.5.1 Single EPR Pair Generation with Strict Resource Reservation . . . 88
3.5.2 Multiple EPR Pair Generation with Strict Resource Reservation . . . 90
3.5.3 Multiple EPR Pair Generation with Resource Relinquishment . . . 91
3.6 Numerical Evaluation . . . 92
3.6.1 Homogenous Traffic . . . 95
3.6.2 Non-homogeneous traffic . . . 98
3.7 Conclusion and Future Work . . . 100
3.8 Appendices . . . 101
3.8.1 Phase-Level System Model . . . 101
3.8.2 Probability of Reaching a Phase in Single EPR Pair Generation Mode 107
3.8.3 Derivation of the Stationary Distribution for Each Service Model . . 108
3.8.4 Proofs . . . 111
3.8.5 Physical EGS Operation Protocols . . . 117
3.8.6 Simulation Implementation . . . 119
3.8.7 Extended Data . . . 120
References . . . 120

4 A Modular Quantum Network Architecture for Integrating Network Scheduling with Local Program Execution 127
4.1 Introduction . . . 127
4.1.1 Structure . . . 129

4.2 Background . . . 129
4.2.1 Network Model . . . 129
4.2.2 Related Work . . . 129
4.3 Design Considerations . . . 131
4.3.1 Network . . . 132
4.3.2 Applications . . . 135
4.4 Network Architecture . . . 137
4.4.1 Periodic Computation and Distribution of Schedules . . . 137
4.4.2 Application Sessions . . . 138
4.4.3 Demand Format . . . 140
4.4.4 Network Capabilities Update . . . 142
4.4.5 Capability Negotiation . . . 143
4.4.6 Demand Registration . . . 143
4.4.7 Session Initialisation . . . 144
4.4.8 Processing Demands and Packet Generation Tasks . . . 144
4.4.9 Network Scheduling . . . 148
4.4.10 Distributing the Network Schedule . . . 150
4.5 Implementation . . . 150
4.5.1 Network Schedule Format . . . 151
4.5.2 Service Model . . . 151
4.5.3 Network Capability Update . . . 151
4.5.4 Capability Negotiation . . . 152
4.5.5 Demand Registration and Queuing . . . 152
4.5.6 Creating Packet Generation Tasks . . . 152
4.5.7 Network Scheduling . . . 152
4.5.8 End Node Scheduling . . . 154
4.6 Evaluation . . . 154
4.6.1 Metrics . . . 155
4.6.2 Network Configuration . . . 156
4.6.3 Admission Control . . . 156
4.6.4 Session Model . . . 157
4.6.5 Results . . . 158
4.7 Conclusions and Future Work . . . 163
4.8 Appendices . . . 165
4.8.1 Notation . . . 165
4.8.2 Who Knows What . . . 165
4.8.3 Methods for Creating Packet Generation Tasks . . . 167
4.8.4 Additional Evaluation Details . . . 168
References . . . 176

5 Arqon: A suite of control applications enabling a reliable quantum network 185
5.1 Introduction . . . 185
5.2 Related Work . . . 187
5.3 Design Considerations . . . 189
5.3.1 Core Considerations . . . 189
5.3.2 Network Model . . . 190

5.3.3 Physical Considerations . . . 192
5.3.4 Timescales . . . 192
5.3.5 Architecture . . . 193
5.4 Arqon Suite of Control Applications . . . 195
5.4.1 Network Manager . . . 197
5.4.2 Demand Manager . . . 199
5.4.3 Network Scheduler . . . 202
5.4.4 Schedule Manager . . . 209
5.5 Performance Analysis . . . 209
5.6 Implementation . . . 211
5.6.1 End Nodes . . . 211
5.6.2 Applications . . . 211
5.6.3 Creating Packet Generation Tasks . . . 212
5.6.4 Network Topology . . . 213
5.6.5 Path Computation . . . 214
5.6.6 Network Capabilities . . . 215
5.6.7 Filling Classes . . . 215
5.6.8 Execution of Network Schedules . . . 215
5.7 Complexity Analysis . . . 215
5.8 Evaluation . . . 217
5.8.1 Service Reliability . . . 217
5.8.2 Additional Performance Metrics . . . 220
5.8.3 Numerical Complexity Analysis . . . 223
5.9 Conclusion and Outlook . . . 227
5.10 Appendices . . . 227
5.10.1 Notation . . . 227
5.10.2 Network Scheduler Algorithms . . . 229
5.10.3 Performance Analysis . . . 234
5.10.4 Complexity Analysis of the Network Scheduler . . . 249
5.10.5 Implementation: Supporting Results and Formal Definitions . . . 259
5.10.6 Code Listings . . . 264
References . . . 277

6 Conclusion **285**
6.1 Summary of Results . . . 285
6.2 Required Hardware for a Metropolitan Hub . . . 287
6.3 Arqon Integration and Experimental Evaluation . . . 291
6.3.1 Components of the Integrated System . . . 291
6.3.2 Test Applications . . . 293
6.3.3 Test Phases . . . 295
References . . . 296

7 Outlook **299**
7.1 Further Protocol Development for Deploying Multi-User Networks . . . 299
7.1.1 Protocols Implementing a Network Capability Update . . . 299
7.1.2 Protocols Implementing Capability Negotiation . . . 300

7.1.3 Configuring Application Demands 300
7.1.4 Entanglement Generation in Heterogeneous Networks 301
7.2 Beyond Centrally Controlled Architectures 301
References 302

Acknowledgments **305**

Curriculum Vitæ **313**

SUMMARY

The internet is a global computer network that enables a vast range of applications, including video streaming, online shopping, and remote access to powerful computing and data storage systems. These applications have transformed the logistics of everyday life for billions of people and deliver quantifiable economic benefits to society.

The internet as it exists today evolved gradually from simpler, smaller-scale networks that originally supported few applications and were accessible to relatively few users. Scaling these classical networks required substantial innovation across many areas, including the development of control architectures that allow multiple users to simultaneously execute applications with diverse and competing requirements.

Quantum computers are an emerging technology that execute applications on a state space defined by quantum mechanical systems, rather than the binary digital state space of classical computers. They can be integrated into hybrid quantum-classical computing nodes capable of supporting both quantum and classical applications. New classes of applications have been identified that require the classical network connecting such nodes to be supplemented by an additional communication layer operating over a quantum state space. A network of hybrid quantum-classical nodes augmented in this way is called a quantum network.

In this thesis, we address the problem of developing control architectures that enable multiple users to simultaneously execute quantum network applications. This is an urgent challenge because steady technological progress on hybrid quantum nodes is bringing testbed quantum networks within practical reach. As these testbeds are constructed, they will need to be operated and opened up to serve applications from multiple users concurrently.

Shared infrastructure reduces the cost and complexity of connecting multiple nodes. In the second and third chapters of this thesis, we introduce metropolitan hubs as a solution for scaling access to quantum networks. These centralized resource-sharing stations provide many nodes with access to the resources required for quantum communication. We develop control architectures for these hubs that define how users request service and specify how those requests are handled. Serving requests requires a metropolitan hub to allocate nodes access to a shared resource. We develop broadly applicable service models and protocols for resolving the competition that arises when many users concurrently request access to shared and limited resources.

Quantum network applications are diverse, imposing a wide range of quantum communication service requirements. In the fourth chapter, we develop a demand format that allows nodes to request service reflecting the nuanced requirements of executing their quantum network applications. Building on this, in the fourth and fifth chapters, we develop a control architecture for long-distance quantum networks. Such networks comprise many shared infrastructural components, including metropolitan hubs and

repeater chains that mitigate losses incurred over long distances. To prevent newly arriving demands from disrupting service to existing ones, we develop an admission control mechanism that accounts for the end-to-end resource contention each demand introduces. To satisfy many accepted demands for service in a limited duration of time, we introduce a method of producing network schedules that repeatedly allocates end-to-end service to each of these demands.

The quantum network control architectures developed in this thesis are designed to be compatible with realistic, modern hybrid quantum-classical nodes. The essential next step towards making multi-user quantum networks a reality is to deploy the architectures developed here in test-bed quantum networks.

SAMENVATTING

Het internet is een wereldwijd computernetwerk dat een breed scala aan toepassingen mogelijk maakt, waaronder videostreaming, online winkelen en toegang op afstand tot krachtige reken- en gegevensopslagsystemen. Deze toepassingen hebben de logistiek van het dagelijks leven voor miljarden mensen veranderd en leveren meetbare economische voordelen op voor de samenleving.

Het internet zoals het vandaag bestaat, is geleidelijk geëvolueerd uit eenvoudigere, kleinschaligere netwerken die oorspronkelijk slechts weinig toepassingen ondersteunden en toegankelijk waren voor relatief weinig gebruikers. Het opschalen van deze klassieke netwerken vereiste aanzienlijke innovatie op vele gebieden, waaronder de ontwikkeling van besturingsarchitecturen die meerdere gebruikers in staat stellen tegelijkertijd toepassingen uit te voeren met uiteenlopende en concurrerende vereisten.

Kwantumcomputers zijn een opkomende technologie die toepassingen uitvoert op een toestandsruimte die wordt gedefinieerd door kwantummechanische systemen, in plaats van de binaire digitale toestandsruimte van klassieke computers. Ze kunnen worden geïntegreerd in hybride kwantum-klassieke rekenknooppunten die zowel kwantum- als klassieke toepassingen kunnen ondersteunen. Er zijn nieuwe klassen toepassingen geïdentificeerd waarvoor het klassieke netwerk dat dergelijke knooppunten verbindt, moet worden aangevuld met een extra communicatielaag die opereert over een kwantumtoestandsruimte. Een netwerk van hybride kwantum-klassieke knooppunten dat op deze manier is uitgebreid, wordt een kwantumnetwerk genoemd.

In dit proefschrift behandelen we het probleem van het ontwikkelen van besturingsarchitecturen die meerdere gebruikers in staat stellen tegelijkertijd kwantumnetwerktoepassingen uit te voeren. Dit is een urgente uitdaging, omdat gestage technologische vooruitgang op het gebied van hybride kwantumknooppunten testbed-kwantumnetwerken binnen praktisch bereik brengt. Naarmate deze testbeds worden gebouwd, zullen ze beheerd moeten worden en opengesteld moeten worden om toepassingen van meerdere gebruikers gelijktijdig te bedienen.

Gedeelde infrastructuur verlaagt de kosten en complexiteit van het verbinden van meerdere knooppunten. In het tweede en derde hoofdstuk van dit proefschrift introduceren we stedelijke hubs als oplossing voor het opschalen van toegang tot kwantumnetwerken. Deze gecentraliseerde stations voor het delen van middelen bieden veel knooppunten toegang tot de benodigde middelen voor kwantumcommunicatie. We ontwikkelen besturingsarchitecturen voor deze hubs die bepalen hoe gebruikers service aanvragen en hoe die verzoeken worden afgehandeld. Het afhandelen van verzoeken vereist dat een stedelijke hub knooppunten toegang verleent tot een gedeelde bron. We ontwikkelen breed toepasbare servicemodellen en protocollen voor het oplossen van de concurrentie die ontstaat wanneer veel gebruikers gelijktijdig toegang vragen tot gedeelde en beperkte middelen.

Kwantumnetwerktoepassingen zijn divers en stellen een breed scala aan vereisten aan kwantumcommunicatiediensten. In het vierde hoofdstuk ontwikkelen we een vraagformaat waarmee knooppunten service kunnen aanvragen die aansluit bij de genuanceerde vereisten voor het uitvoeren van hun kwantumnetwerktoepassingen. Voortbouwend hierop ontwikkelen we in het vierde en vijfde hoofdstuk een besturingsarchitectuur voor langeafstandskwantumnetwerken. Dergelijke netwerken omvatten veel gedeelde infrastructurele componenten, waaronder stedelijke hubs en herhalerkettingen die verliezen over grote afstanden beperken. Om te voorkomen dat nieuw binnenkomende verzoeken de dienstverlening aan bestaande verzoeken verstoren, ontwikkelen we een toelatingsbeheersmechanisme dat rekening houdt met de end-to-end bronnencontentie die elk verzoek introduceert. Om aan veel geaccepteerde dienstverzoeken te voldoen binnen een beperkte tijdsduur, introduceren we een methode voor het opstellen van netwerkschema's die herhaaldelijk end-to-end service toewijst aan elk van deze verzoeken.

De in dit proefschrift ontwikkelde kwantumnetwerk-besturingsarchitecturen zijn ontworpen om compatibel te zijn met realistische, moderne hybride kwantum-klassieke knooppunten. De essentiële volgende stap naar het realiseren van kwantumnetwerken voor meerdere gebruikers is het inzetten van de hier ontwikkelde architecturen in testbedkwantumnetwerken.

List of Publications

Included in this Thesis

Journal Articles

13. T. R. Beauchamp, H. Jirovská, S. Gauthier, and S. Wehner. *A Modular Quantum Network Architecture for Integrating Network Scheduling with Local Program Execution.* IEEE Transactions on Quantum Engineering, 2026, vol. 7, pp. 4100629-4100629. Art no. 4100629.

12. S. Gauthier, T. Vasantam, and G. Vardoyan. *On-Demand Resource Allocation for A Quantum Network Hub.* IEEE Transactions on Quantum Engineering, 2026, vol. 7, pages 1–30. Art no. 4100330

11. S. Gauthier, G. Vardoyan, and S. Wehner. *An Architecture for Control of Entanglement Generation Switches in Quantum Networks.* IEEE Transactions on Quantum Engineering, 2023, vol. 4, pages 1–17. Art no. 4100717.

Pre-print Articles

10. S. Gauthier[†], T. R. Beauchamp[†], and S. Wehner. *Arqon: a Suite of Control Applications Enabling a Reliable Quantum Network.* arXiv:2604.08692, 2026.

Conference Papers

9. S. Gauthier[†], T. R. Beauchamp[†], and S. Wehner. *Arqon suite of quantum network control applications.* Proceedings of the 2nd Workshop on Quantum Networks and Distributed Quantum Computing (QuNet '25), 2025, pages 49–52.

8. T. R. Beauchamp, H. Jirovská, S. Gauthier and S. Wehner. *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), 2025, pages 1–7.

7. S. Gauthier, T. Vasantam, and G. Vardoyan. *An on-demand Resource Allocation Algorithm for a Quantum Network Hub and its Performance Analysis.* 2024 IEEE International Conference on Quantum Computing and Engineering (QCE), 2024, vol. 01, pages 1748–1759.

6. S. Gauthier, G. Vardoyan, and S. Wehner. *A Control Architecture for Entanglement Generation Switches in Quantum Networks.* Proceedings of the 1st Workshop on Quantum Networks and Distributed Quantum Computing (QuNet '23), 2023, pages 38–44.

[†]Equally contributing authors.

NOT INCLUDED IN THIS THESIS

WHITE PAPERS

5. T. R. Beauchamp, S. Gauthier, and S. Wehner. *White Paper on Quantum Internet Computer Science Research Challenges.* arXiv:2511.16745, 2025.

JOURNAL ARTICLES

4. H. A. Atikian, N. Sinclair, P. Latawiec, X. Xiong, S. Meesala, S. Gauthier, D. Wintz, J. Randi, D. Bernot, S. DeFrances, J. Thomas, M. Roman, S. Durrant, F. Capasso, and M. Lončar. *Diamond mirrors for high-power continuous-wave lasers.* Nature Communications, 2022, vol. 13, no. 1.

3. B. Machielse, S. Bogdanovic, S. Meesala, S. Gauthier, M. J. Burek, G. Joe, M. Chalupnik, Y. I. Sohn, J. Holzgrafe, R. E. Evans, C. Chia, H. Atikian, M. K. Bhaskar, D. D. Sukachev, L. Shao, S. Maity, M. D. Lukin, and M. Lončar. *Quantum Interference of Electromechanically Stabilized Emitters in Nanophotonic Devices.* Physical Review X, 2019, vol. 9, no. 3.

CONFERENCE PAPERS

2. J. Smith, T. R. Beauchamp, S. Gauthier, O. Bouchmal, S. Wehner. *Tools for Reducing Service Time in Near-Term Quantum Networks.* IEEE International Conference on Quantum Computing and Engineering—QCE26, (Accepted). 2026.

1. T. R. Beauchamp, A. Sebastián-Lombraña, S. Gauthier, J. J. Romero, V. Martin, S. Wehner, and L. Ortiz. *Towards a common framework for quantum information networking.* Proceedings of the 2nd Workshop on Quantum Networks and Distributed Quantum Computing (QuNet '25), 2025, pages 56–59.

LIST OF PATENTS

NOT INCLUDED IN THIS THESIS

PATENTS

2. S. Gauthier, A. J. Stolk, M. Iuliano, S. Wehner, I. te Raa-Derckx, and R. Hanson. *A method for real-time state-validation of entanglement in an extendible quantum network*. Patent application, December 20, 2024.

1. S. Gauthier, G. Vardoyan, and S. Wehner. *Method for controlling service demand of quantum entanglement generation in quantum network*. Patent application, November 4, 2022. Granted, May 24, 2024.

1

INTRODUCTION

Although in many instances certain technological advancements can indeed be accounted for in terms of physics or chemistry, in other words, can be seen as based on pure science, it should not be overlooked that the problem was originally not cognitive but technical. With an eye to solving a technical problem, we undertake inquiries into what is called pure science. Our procedures are extremely selective. Out of infinitely many possible channels of research only very few are chosen. Problems thus are investigated not *with an eye to increasing knowledge but with an eye to a solution of a technical problem.*

Henryk Skolimowski

1.1. THE PROMISE OF QUANTUM NETWORKS

A technology is a tool, method, or system that enables new capabilities or improves upon existing ones. This improvement may take various forms, including enabling tasks that were previously impossible, performing existing tasks more efficiently, reducing costs, or improving reliability or accessibility. In other words, a technology either delivers new functionalities that do not otherwise exist, or provides advantages over existing approaches in dimensions such as performance, cost, scalability, or usability [1, 2]. Any specific technology may be applied in a variety of different use cases, and these use cases generally deliver outcomes that are beneficial to the user.

Quantum networks are a category of developing technologies for which several beneficial uses have been identified. In the domain of security and privacy, a quantum network can for example be used to achieve secure communication [3, 4], secure authentication of users or messages [5, 6], (verifiably [7]) secure cloud computations [8–10], and position verification [11, 12]. In the domain of logistics and infrastructure, a quantum network can be used to achieve improved load balancing of a network – such as a power grid or the back-end of a web-service – [13, 14], to reduce communication requirements [15],

to improve distributed control – for example of a network of computers – [16, 17], to enable more accurate clock synchronization [18], and to extend the baselines of telescopes [19]. Novel use cases of a quantum network include proof of deletion [20], and quantum money [21]. Moreover, technological advancement of quantum networks can be directly related to efforts to scale up quantum computers. Modern (classical) computers typically comprise a network of individual modules, such as the multiple cores that make up a single CPU [22], or the multiple networked computers which compose a data center [23] or high-performance computer [24]. The success of scaling computers using networking methods motivates the use of a quantum network to achieve scalable quantum computing [25–27], where the connection of multiple quantum computers to form a network of quantum computers can enable execution of large computations which are too complex to be executed on any of the quantum computers individually. This use case appears in the road-maps of many major companies which are working to commercialize quantum computers, as an essential step for scaling up their systems [28–30].

These possible use cases for quantum networks are not exhaustive, meaning that additional use cases may also be identified. They do however illustrate that a quantum network is a very promising type of technology, which has the potential to meaningfully impact many sectors of industry, infrastructure, and fundamental science.

1.2. Quantum Information

Having established that there are many promising use cases for quantum networks, it is worth asking whether these applications truly require quantum networks, or whether they could be realized using established classical technologies.

More specifically: *Can classical computer networks, quantum computers connected only by classical communication, or hybrid classical-quantum approaches sufficiently realize the quantum network use cases described earlier?*

The answer requires some context. Suppose that a small number of users simply want to execute one of the earlier identified use cases a single time, or rather infrequently. For some use cases, classical workarounds exist in this scenario. Achieving secure communication provides an illustrative example: there is always a possibility of physically transporting unique cryptographic keys, enabling secure communication over a variety of media and not limited only to computer networks [31]. Another consideration is that classical computers can simulate quantum systems with small to medium complexity, making it possible to emulate certain use cases [32–34]. However, three fundamental properties of quantum information reveal why classical emulation becomes either theoretically impossible or practically infeasible when scaled to realistic network scenarios with many users, users separated by long distances, when executing use cases at high rates, or use cases involving more complex quantum computations. These properties are: the representation of quantum information, the nature of quantum correlations (entanglement), and the no-cloning theorem.

1.2.1. Fundamentals and Representation

The fundamental unit of classical information is a *bit* [35, 36]. It is a symbolic representation of a unit of information, which can take one of two values, 0 or 1. A bit string of length

N is a sequence of N bits, and it can encode exactly N units of information. When paired with a set of methods for interpreting the values of bits in a bit string, or of interpreting entire bit strings, any piece of classical information can be represented by a bit string of sufficient length. In complement, the fundamental unit of quantum information is a qubit [37]. Physically, a qubit is a two level quantum system. Unlike a classical bit which is always either 0 or 1, a qubit can be in a superposition state $|\psi\rangle = a|0\rangle + b|1\rangle$, in which complex numbers a and b describe the superposition of the qubit between $|0\rangle$ and $|1\rangle$ [38–40]. Measurement of a qubit always returns either outcome 0 or 1, but the relative likelihood of obtaining outcome 0 or outcome 1 in a measurement is determined by the (superposition) state of the qubit [38, 41]. The 2 complex numbers a and b can be used to translate a qubit of quantum information into a representation using classical information [37, 38]. A normalization condition on quantum states reduces this requirement to 1 complex number to represent a single qubit [40, 41]. In modern computers, a single complex number is usually represented using two bit strings, each of either 32 or 64 bits [42]. Hence, communication of a classical representation of an arbitrary qubit typically requires communicating 64 or 128 bits. In comparison, using a quantum network, it is possible to communicate an arbitrary qubit by sending 1 unit of quantum information and 2 bits of classical information [43, 44]. Similarly, a representation of an arbitrary string of N qubits using classical information requires $2^N - 1$ complex numbers, where the -1 is due to the normalization condition on quantum states. Thus, there is an exponential overhead in the number of bits that need to be communicated if classical information is used to represent arbitrary strings of quantum information. This exponential overhead is one of the characteristic motivations for employing quantum networks to communicate quantum information. Typically at the end of a quantum network application, or at strategic points during the execution of such an application, a quantum state will be measured. Measurement of a string of N qubits extracts an N bit string of classical information from the quantum state in the form of N measurement outcomes. The pattern of these outcomes is determined by the quantum state and the type of measurements made [38].

1.2.2. CORRELATIONS AND ENTANGLEMENT

Another and even more fundamental motivation is that many quantum network applications, including [7–9, 15, 18–21], are not possible using classical representations of quantum information. This is because applications of quantum networks leverage a distinct aspect of quantum systems: they can share correlations that are classically impossible [45–47], as demonstrated experimentally through the violation of Bell inequalities [48, 49]. A correlation is a type of relationship between systems where the state or characteristics of one system has some dependence on the state or characteristics of one or more other systems. The strongest form that distinctly quantum correlations can take is called entanglement [45, 46]. A bipartite (between two qubits) quantum state is said to

be maximally entangled if it corresponds to one of the following states,

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}\big[\,|00\rangle + |11\rangle\,\big] \tag{1.1}$$

$$|\Phi^-\rangle = \frac{1}{\sqrt{2}}\big[\,|00\rangle - |11\rangle\,\big] \tag{1.2}$$

$$|\Psi^+\rangle = \frac{1}{\sqrt{2}}\big[\,|01\rangle + |10\rangle\,\big] \tag{1.3}$$

$$|\Psi^-\rangle = \frac{1}{\sqrt{2}}\big[\,|01\rangle - |10\rangle\,\big]. \tag{1.4}$$

Independent measurements of the qubits of states (1.1) or (1.2) will always return identical outcomes, either both zero or both one. The outcomes of these measurements are said to be identically correlated. Similarly, independent measurements of the qubits of a state (1.3) or (1.4) will always return opposite measurement outcomes. The outcomes of these measurements are said to be anti-correlated. These correlations can be observed regardless of the distance between the location of the two qubits or differences in the times at which they are measured. In this way, maximally entangled states enable perfect instantaneous coordination. If two physically separated parties each have one qubit of a known maximally entangled state (1.1), (1.2), (1.3) or (1.4), then either party can measure their state and instantly learn what outcome the other party will measure or has already measured.

In the terminology of a quantum network, we can refer to bipartite entanglement as an entangled link. In the process of preparing an entangled link, errors (also known as noise processes) can occur. The result is a noisy entangled link which corresponds to a state that is not maximally entangled. A quality metric called the fidelity can be used to quantify how useful a noisy entangled link is for enabling an application that is designed to consume maximally entangled links [50, 51]. The fidelity indicates the mathematical overlap between a general two qubit state ρ and a target maximally entangled state $|\Psi\rangle$ and is denoted $F(\rho, |\Psi\rangle)$, or simply as F. Entangled links and the classically impossible correlations which can be derived from them are the fundamental resources that enable quantum network applications.

1.2.3. No-Cloning

There is a final fundamental distinction between quantum information and classical representations of that information. Classical information can be copied arbitrarily. A consequence is that an entity with access to a piece of classical information can copy it, and potentially use it in a way that was not originally intended by the owner of the information. In contrast, arbitrary qubits and arbitrary strings of qubits are subject to a *no-cloning* [52, 53] property, which means that they cannot be perfectly copied. Moreover, an attempt to learn something about a qubit or string of qubits changes it, possibly allowing the owner to detect that an attempt was made to observe the state [21, 54]. This property is the foundation of security based applications of quantum networks, such as secure communication [3, 4].

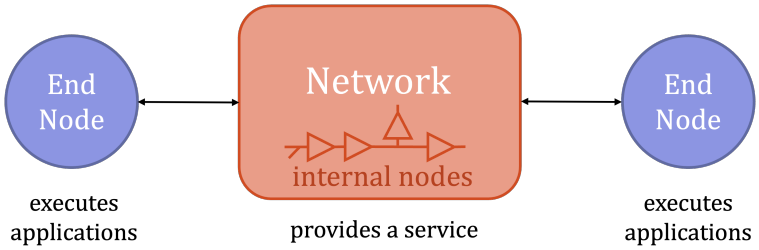


Figure 1.1: A quantum network provides a service to end nodes, supporting the execution of applications. The network may have a complex topology and include many types of internal nodes, but the end nodes interact with it as a black box system delivering a service.

1.3. QUANTUM NETWORKS AS SERVICE PROVIDERS

A quantum network consists of two distinct categories of components. First, there are independent systems that can cooperate to execute quantum network applications based on user input; these are called end nodes. Second, there are internal components of the network, which are controlled by sets of rules called protocols. Network protocols coordinate the actions of the internal components to provide service to sets of end nodes. The service provided is either the distribution of a classical or quantum information resource to sets of end nodes, or alternatively the opportunity to make use of the network components to generate such resources. This model of quantum networks as service providers to end nodes is illustrated in Figure 1.1.

Applications realize use cases and consist of one or more programs at one or more end nodes [55]. In a network setting, an application typically comprises two programs at two different end nodes. These programs contain both classical instructions (communication and computation) as well as quantum instructions (communication and computation).

From the user perspective, the internal components of the network may be viewed as a black-box system which is capable of delivering service. End nodes interact with the black-box system by demanding service, to be delivered before a specified deadline, and possibly including further application specific instructions. The black-box of the network responds to demands by providing service with some quality and frequency which may depend on the specific instructions included in a demand.

1.4. ENTANGLEMENT GENERATION

1.4.1. METHODS

Over short distances or within a single node, entangled links can often be generated between two qubits by the application of a quantum logic operation known as a *two-qubit gate* [56, 57]. Quantum logic operations leverage the physical degrees of freedom of qubits and how these degrees of freedom can be made to interact with each other. An entangling two-qubit gate can be understood as turning on an interaction coupling the states of the two qubits and resulting in entanglement, and subsequently turning the interaction off again. Two-qubit gates are often based on electromagnetic pulses, such as microwave pulses [58–60] that directly drive qubit transitions, or optical pulses [61] that couple qubits through shared degrees of freedom such as the motional modes of ions in

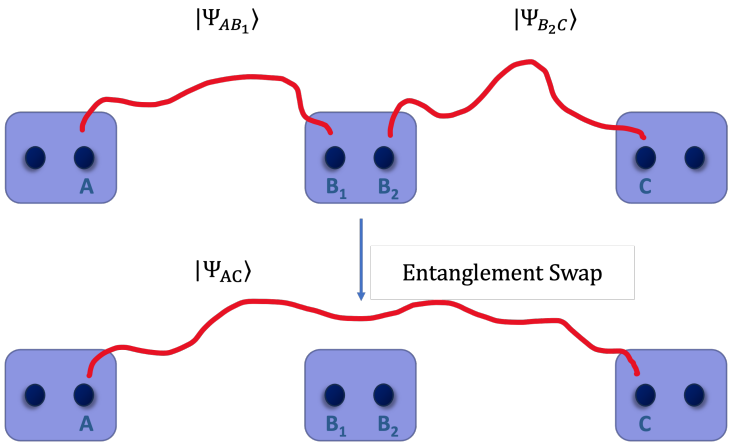


Figure 1.2: A simple quantum network with end nodes A and C wishing to share entanglement, and an intermediate node B with two qubits B_1 and B_2, assisting them with the task. Initially, two entangled links are established: $|\Psi_{AB_1}\rangle$ between A and B_1 and $|\Psi_{B_2C}\rangle$ between B_2 and C. Node B then performs a swapping operation to directly entangle A and C's qubits. Depending on the distance between A and C, direct generation of entanglement (without an intermediate node), may not be feasible.

ion traps. The implementation of these two-qubit gates is limited to short distances, as the physical interactions mediating entanglement (such as electromagnetic coupling or shared motional modes) decrease rapidly with qubit separation, necessitating alternative approaches for generating entanglement over longer distances.

The generation of entangled links between physically separated quantum nodes is, in current and near-term implementations, a probabilistic process [62–64], though deterministic schemes are theoretically possible with methods supporting high enough attempt rates and sufficient error correction capabilities [65, 66]. Link generation can be attempted according to many distinct methods [63, 64]. These methods typically rely on photons as intermediaries: either the state of a photon is entangled with the state of one or more nodes, or two entangled photons are produced at a midpoint between nodes. The photons are able to travel between nodes through mediums such as optical fiber or free space, and transfer procedures called entanglement swaps are used to trade node-photon or photon-photon entanglement for node-node entanglement. Sets of components external to nodes that enable these methods, such as Bell-State Analyzers (BSAs) [67–69] or entangled photon sources [70, 71], may be collectively called entanglement generation interfaces.

To facilitate the execution of quantum network applications, it is useful to have a method of determining whether an attempt to generate an entangled link succeeds or fails, without resorting to measurement of the quantum states of the involved systems (which would consume an entangled link, rendering it useless). So-called *heralded* entanglement generation protocols return success/failure outcomes following each attempt to generate an entangled link [72, 73]. Such success/failure outcomes offer some predictability, allowing systems to trigger the execution of applications requiring entanglement based on it's

availability.

1.4.2. BRIDGING LONG DISTANCES

The generation of an entangled link between two nodes is limited by the distance between them [74]. For example, the noise affecting a photon traveling through an optical fiber grows exponentially with the length of the fiber [75]. As a consequence of the no-cloning theorem, the communication of quantum states cannot be amplified according to traditional methods of signal amplification, which rely on copying data. To overcome the noise (especially loss) incurred in generating end-to-end links over long distances, a strategy is required which relies on the generation of entangled links over shorter distances [75]. A suitable strategy is to generate entangled links between each pair of neighboring nodes on a path connecting two end nodes and to iteratively increase the distance covered by a single entangled link by executing entanglement swaps. This strategy is illustrated in Figure 1.2, where the shorter distance entangled links $|\Psi_{AB_1}\rangle$ and $|\Psi_{B_2C}\rangle$ are swapped for a longer distance link $|\Psi_{AC}\rangle$. A linear chain of internal nodes that is dedicated to bridging long distances by entanglement swapping is known as a repeater chain [75]. Depending on their implementation, entanglement swaps may succeed either deterministically [76], or probabilistically [73, 77]. In either case, generation of end-to-end entangled links is a probabilistic process (barring the realization of deterministic strategies) that succeeds with a lower success rate than that of any of the direct links. In any period of time, entanglement generation may fail over some of the links on a path between end nodes. Depending on the capabilities of the nodes, those links that are generated successfully may be stored in quantum memories while the failed links are re-attempted.

1.4.3. STORAGE OF ENTANGLED LINKS

Quantum systems are subject to time dependent errors due to interactions with their environments [78, 79]. As a consequence, quantum memories, which can be used to store quantum states [80–82], are associated with maximum storage times. After the storage time of a memory elapses a stored state is likely to have significant error with respect to the initially stored state, possibly rendering it useless for further applications. As an added layer of complexity, in some implementations the process of retrieving a stored state from memory can introduce an extra source of error, and precautions need to be taken to minimize these effects [83–85]. Moreover, when a node attempts to generate an entangled link while storing an earlier generated entangled link in memory, the rate at which errors affect the stored entanglement can increase [76, 86].

1.5. TYPES OF QUANTUM NETWORK

A quantum network is any system in which multiple quantum computers or simple quantum devices purposefully share quantum information. Specific types of quantum networks have reached different levels of technological maturity, enable distinct use-cases, and have distinct requirements in order to function correctly.

1.5.1. MODALITIES

TRUSTED REPEATER NETWORKS

A trusted repeater network provides service to end nodes by coordinating the generation of a classical information resource, such as a shared secret key between two end nodes which can be used for encryption of messages sent between the nodes. The classical resource in a trusted repeater quantum network is generated from the outcome of a quantum network application executed separately over direct links. One such application is Quantum Key Distribution (QKD) [3, 4], which enables secure communications.

In such a network, there are two types of nodes: end nodes and trusted repeaters, and quantum information is only shared over direct links between neighboring nodes [87]. That is, quantum information is either shared between an end node and a trusted repeater node, or between two trusted repeater nodes, but not between two end nodes. The trusted repeater nodes process quantum information that they share with other nodes and convert it into a classical information resource. The classical information resources are then communicated in a hop-by-hop manner between trusted repeaters, and from trusted repeaters to end nodes. The outcome is that end nodes share a classical information resource. As classical information can be copied, the end nodes must trust that the repeaters do not collaborate to make a copy of the classical information that they handle, or leak information about it to a malicious entity. If these actions were taken, it would degrade the value to the end nodes of the resulting classical information resource, for example due to compromised security.

Several examples of trusted repeater networks have been deployed and operated under realistic operating conditions [88–90]. Trusted repeater network deployments practically demonstrate the potential of quantum-enhanced secure communications and they prove that quantum information networking is feasible. A fundamental limitation is that quantum network applications cannot be executed between the end nodes of a trusted repeater network.

ENTANGLEMENT-BASED NETWORKS

An entanglement-based network provides service to end nodes by coordinating the generation of end-to-end entangled links. To accomplish this, internal nodes of the network generate direct links and perform entanglement swaps on along an end-to-end route, eventually requiring direct links with end nodes. The resulting end-to-end entangled links allow end nodes to communicate quantum information, without converting it to classical information.

The rate at which entangled links can be generated and the amount of time they can be stored for present major technological challenges to the development of these networks. For example, leading hardware demonstrations based on trapped ion quantum processing nodes have reported entanglement generation rates over a 230 m link of one low fidelity (F=0.58) pair every 2.3 seconds, or one higher fidelity (F=0.83-0.88) pair every 19.4 seconds [91]. Further demonstrations with this hardware have reported storage times of 62 ms [92]. The benchmark for these capabilities in long-distance operating conditions-including fully independently controlled end nodes, use of preexisting deployed optical fiber, and frequency conversion to telecom wavelengths-has been established over a 25 km fiber link (10 km node separation) with the NV color center in diamond [93]. There, an entanglement generation rate of one low fidelity (F=0.524) pair every 46.2 seconds

was demonstrated [93]. In this hardware platform buffer lifetimes of 11.6 ms [76] - 13 ms have been reported [94]. There is a tension between the entanglement generation rates and storage times in these examples, as the expected time to generate an entangled link exceeds the storage time, leading to a low probability that the links required to enable entanglement swaps co-exist.

The components of entanglement-based networks must sufficiently address these challenges to enable end nodes to execute quantum network applications. Moreover, the methods by which a network coordinates the generation of end-to-end entangled links must incorporate management strategies for the effects of these technological considerations.

In principle, all quantum network applications can be executed between the end nodes of an entanglement-based network.

1.5.2. APPLICATION PROGRAMMING PARADIGMS

Quantum network applications require multiple end nodes to collaborate. An application consists of a set of programs, one program for each end node that collaborates to execute the application. A program specifies the instructions that an end node must execute in order to fulfill their role in completing the application. A program may be organized into blocks of instructions, and it may indicate which blocks are possible to execute at the same time, and which must be executed in a specific order. A program will typically also include logic specifying how to handle failures. An example is that a program may specify re-attempting an instruction until the first time it succeeds.

In the simple quantum network model introduced above, it was assumed that end nodes are independent. To enable a more complete overview of the possible types of quantum networks, let us first temporarily relax this assumption, and then define it more precisely.

INDEPENDENT END NODES

End nodes are independent if they locally determine the programs which they will execute, as part of a quantum network application. Independent end nodes may collaborate in advance of beginning an application to negotiate details of how they will locally configure their programs. During application runtime the end nodes can exchange messages about the outcomes of executed instructions.

Other than what is communicated in the form of classical messages or what is learned by measurement of shared quantum information, independent end nodes do not have any insight into the progress of the application program execution on the other end nodes.

COLLECTIVELY CONTROLLED END NODES

A set of end nodes is collectively controlled if there is a common entity which determines the program that each end node executes as part of a quantum network application. During the program execution on each end node, the collective controller may monitor the complete activities of each end node and all measurement outcomes. The collective controller may leverage information about the status of the program execution on each end node to modify the programs during application runtime.

This approach offers potential benefits to application performance, such as greater coordination between end nodes and efficiency in terms of the total number of instructions executed to complete the application. Certain quantum network use cases, such as scaling quantum computers may primarily rely on this application programming paradigm [27, 95–97].

Collective control of end nodes is not suitable for every application of a quantum network. For example, applications which involve an element of blindness as a security consideration, such as [7–9] or pertain to distributed consensus, such as [16, 17] may explicitly require that end nodes are independent.

1.5.3. A Classification of Entanglement-Based Networks

For quantum network applications, communication time between nodes is more relevant than physical distance. Classical communication is required to coordinate the actions of internal nodes of the network, enabling service delivery to end nodes. Hence the classical communication time limits the degree of coordination that is achievable. Quantum network applications generally require the exchange of both quantum and classical information during the application [55, 94]. Often, application success requires communication times to be less than the relevant timescales of quantum devices, such as quantum memory lifetimes. While the minimum communication time is bounded by the speed of light through the transmission medium [98, 99], actual communication times in modern (classical) networks often significantly exceed this physical limit due to routing, processing, and congestion delays [22, 23, 100]. Communication time may be considered long if it greatly exceeds the duration of time required to execute the shortest program instruction, and short if it is comparable to or less than this instruction execution time.

A quantum network can be described as robust against N hardware failures if the failure of N internal nodes of the network does not cause an interruption to the successful delivery of service by the network. Similarly, a quantum network may be described as reliable if accepted demands are satisfied with a high probability and an increase in the quantity or frequency of demands for service by user controlled end nodes does not result in a substantial decrease to the quality with which the network serves demands. A decrease in service quality is substantial if it prevents the successful execution of a quantum network application before it's deadline.

Network-on-chip Systems and Modular Computers

Network-on-chip (NOC) systems such as [101, 102] or modular computers such as [103, 104] are quantum networks over short distance scales. In an NOC system, it may be the case that the average communication time between two nodes is short (sub-nanoseconds (ns) to microseconds (μs)) and the standard deviation of communication times over all node pairings within the network is insubstantial (ns). In a modular computer the average communication time between two nodes may exceed that of an NOC system, although remain short (ns to milliseconds (ms)), but the variance of communication times may be greater than in NOC systems (ns to ms).

An NOC system or modular quantum computer is most compatible with an application programming paradigm in which all nodes, including end nodes, are collectively controlled. In these networks there need not be a fundamental distinction between end

nodes of the network and the internal nodes, as these roles may actually be re-assigned by a collective controller in every application program. For example, end nodes may be temporarily assigned by partitioning the logical qubits of a modular quantum computer. These systems may however, act as a single end or internal node within the context of a *long-distance quantum network* or a *quantum internet*. In those cases, the entire NOC or modular computer, or a partition thereof may constitute an independent end node of the larger network.

LONG-DISTANCE QUANTUM NETWORKS

A long distance quantum network enables applications which require end nodes to be independent or physically distant from each other in order to unlock their full potential. These include applications in reducing communication requirements [15], distributed consensus [16, 17], and many security applications [3–9]. A long distance network is needed to enable transmission of information between quantum computers or devices in distant locations, such as between two R&D centers of one company that are located in different cities, or between a simple client device and an advanced quantum processor that can offers cloud-based quantum computation.

Over distance scales in the tens of kilometers, communication times range from tens of μs to ms. Over hundreds or thousands of kilometers, communication times are greater than hundreds of μs and may exceed tens or hundreds of ms. The standard deviation in communication times in such a network may also be very substantial (tens of ms). A challenge in the design of methods for successfully operating long distance quantum networks is to account for long and highly variable communication times.

A long distance quantum network is most compatible with an application programming paradigm in which all end nodes are independent. The internal nodes of a long distance network may be either collectively controlled, such as in a centrally controlled network, or they may make use of distributed methods, such as decision consensus or traffic monitoring methods to achieve effective independent control.

A QUANTUM INTERNET

The term *internet*, coined in [105] is a shortening of the term *internetwork*, referring to an interconnected network composed of several sub-networks. In general, sub-networks may be independently administrated and may employ different methods of serving demands and creating alignment between components. Together with the word internet, ref. [105] introduced a standard method (Transmission Control Protocol) of addressing challenges that uniquely arise in the context of an internetwork. These include the possibility of differences in the semantic formats of messages and data structures used in different sub-networks, differences in systems of addressing the source and destination of a service request, and the need for gateways implementing a standard interface between sub-networks.

For a quantum network to qualify as a quantum internet, it must satisfy two categories of requirements. First, it should be capable of serving end-to-end requests originating from separately administered sub-networks, analogous to the classical internetwork model. Second, since the modern Internet has evolved into a highly advanced computer network [100] that is deeply integrated into society [106, 107], the term internet has become associated with specific quality expectations: robustness [108], reliability [100, 109],

high communication rates [110], and versatile application support [100, 111]. Therefore, validation that a quantum network exhibits these qualities-robustness, reliability, high quantum information communication rates, and support for diverse applications-should also be prerequisites for qualification as a quantum internet.

1.6. Protocols and Architectures

1.6.1. Protocols

A protocol is a set of rules governing the actions of a system. Protocols serve two primary purposes: ensuring predictable behavior from each component of a system and coordinating interactions between different components. This coordination creates alignment across a system and may encompass the timing and ordering of actions, resource allocation, and semantic agreement about exchanged information. Complex protocols often contain sub-protocols, which are specialized rule sets that govern smaller components or more specific actions.

A lack of alignment between protocols may lead to undesirable effects for the system they govern. In the context of a quantum network, a lack of protocol alignment could, for example, result in a persistent situation in which the service delivered by the network does not satisfy the demands made by end nodes.

1.6.2. Architectures

An architecture is a plan for a system. It defines the components of the system, the responsibilities of those components, and the interfaces that connect individual components. For illustration, the architecture of a building is a complete blueprint, which includes the foundation and structural support of the building, the plans for the individual rooms, the placement and size of the doorways which connect the rooms, the placements and size of windows which connect the building to the outside world, and the design considerations which influence the utility of the building to its users.

Everything about an architecture should reflect it's intended function. This means that the starting point for developing an architecture is identification of the requirements or design considerations that must be satisfied in order to deliver this functionality. To achieve its function, an architecture must be implemented and then put to use. How it is put to use influences the degree to which it satisfies its function. For instance, a library better fulfills its purpose when users follow protocols that maintain a quiet environment by prohibiting activities like shouting. Architectures work best when accompanied by clear usage protocols, enabling them to achieve their function. It is practical to co-develop usage protocols during the design phase of an architecture, to ensure the architecture can actually deliver its intended function.

An architecture which is instantiated and equipped with a set of protocols is called an implementation of the architecture. The success of an implementation can be assessed by comparing the functionality delivered by the implementation to the requirements or design considerations which informed the intended functionality.

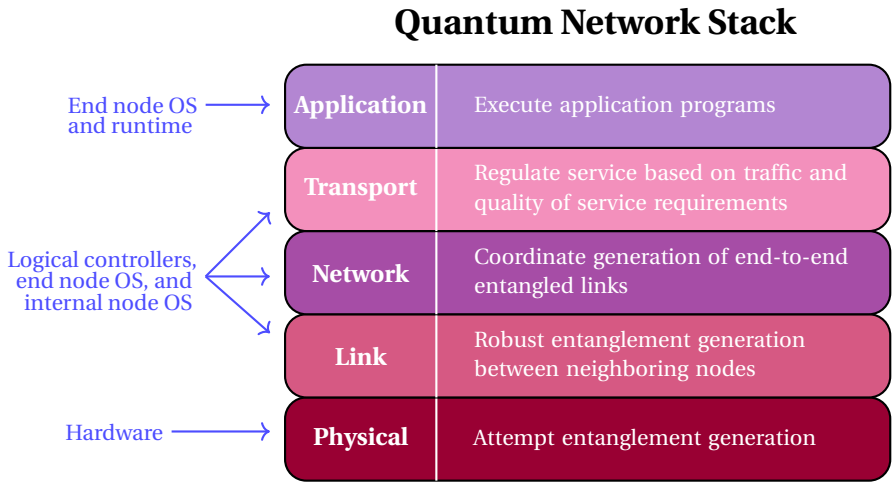


Figure 1.3: Functional allocation in a quantum network protocol stack. A full stack quantum network includes an implementation of each layer and enables the execution of user selected applications on end nodes.

1.6.3. NETWORK PROTOCOL STACK

Network protocols can be grouped into functional layers known as a network stack [112, 113], in which each layer exposes a service to the layer above, packaged such that higher layers need not be exposed to implementation details of protocols from lower layers. The quantum network stack considered in this thesis is illustrated in Figure 1.3. The lowest layer is the physical layer, which consists of quantum devices, classical enabling hardware, and the instructions which actuate these devices. It exposes the service of attempting entanglement generation between neighboring nodes to the link layer. Management of entanglement generation attempts is handled by the link layer, which transforms entanglement generation between neighboring nodes into a robust service, to be consumed by the network layer. The network layer coordinates the generation of end-to-end entangled links, consuming the direct links established by the link layer. It exposes these end-to-end links as a service to the transport layer, which regulates demands for end-to-end links and ensures that applications receive an appropriate quality of service. The application layer, hosted on end nodes, consumes end-to-end entangled links to execute their quantum network applications. Logical control elements of internal nodes in a network, possibly including a central controller, cooperate with end node OSs to implement the link and network layers. The transport layer requires the cooperation of end nodes local runtime environments and logical controllers of internal nodes in a network. The application layer is special in that the operating systems and local runtime environments implement the services at this layer, rather than internal network elements.

In Chapters 2 and 3 we will consider network models in which all end nodes are neighboring nodes, hence the protocol stack of these networks includes only a minimal implementation of a network layer. In contrast, in Chapters 4 and 5 we will consider a model of a centrally controlled long distance entanglement-based network. There, the protocol stack includes an extensive network layer, but does not include a transport

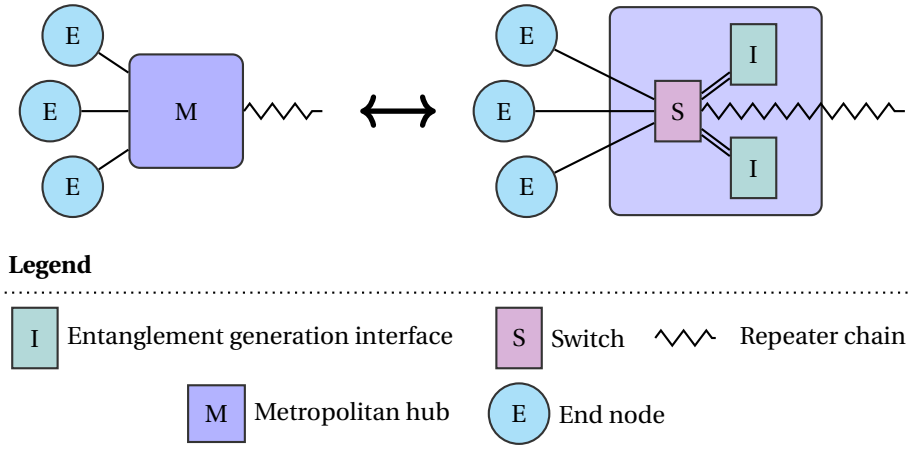


Figure 1.4: Illustration of a metropolitan hub enabling multiple end nodes to share access to a pool of entanglement generation interfaces and a repeater chain. As with pairs of end nodes, entanglement between the repeater chain and an end node is mediated by access to an entanglement generation interface.

layer. In that model, services which are often associated with the transport layer are implemented directly at the network layer.

1.6.4. Quantum Network Control Architectures

A control architecture for a quantum network is a plan for the network which defines types of quantum network nodes, the interfaces through which nodes interact, and a definition of the network layer and/or transport layers of the network stack. In this thesis, control architectures are proposed together with an example protocol or suite of protocols that are compatible with the architecture.

1.7. Enabling Multi-User Access

Deployed quantum networks must facilitate multi-user access, in which multiple sets of end nodes are simultaneously able to successfully execute applications by consuming network services [25, 26]. For such multi-user access to be possible, many end nodes must be able to join the network. When a large number of end nodes are considered, it becomes increasingly impractical to maintain dedicated infrastructure such as optical fiber connections and entanglement generation interfaces between every pair of end nodes. A solution is to build metropolitan hubs – central resource sharing hubs that enable multiple nodes to share the intermediate devices, significantly reducing the complexity and total resources required for large-scale deployment. Metropolitan hubs can be used in the setting of a star-topology metropolitan network, where the only nodes are end nodes and the metropolitan hub. More generally, a metropolitan hub can be a component of a larger network, in which case the metropolitan hub will have a connection to a repeater chain or memory equipped internal network node, in addition to end nodes. A scenario where a metropolitan hub is a component of a larger network is illustrated in Figure 1.4.

Multi-user access requires the network to process simultaneous demands for service.

Multiple demands for shared and limited resources (e.g. the entanglement generation interfaces, repeater chains, and memory equipped internal nodes identified as critical infrastructure) leads to contention for these resources. In general, the total demand may exceed the available resources, resulting in a need for resource allocation protocols producing schedules allowing resource access to be time shared to serve multiple demands. Both a protocol for scheduling resource access and a protocol for determining which demands to serve in what order are key features of a quantum network control architecture.

1.8. SCALING FROM DEMONSTRATIONS TO DEPLOYMENTS

Several generations of experiments have now demonstrated the foundational elements of entanglement-based quantum networks [76, 85, 91–94, 114–116]. Controllable quantum devices have been developed which are capable of generating heralded entanglement with other devices [91, 117–122], manipulating arbitrary quantum states of the device [123, 124], and storing arbitrary quantum states in memory, including entangled states [92, 123, 125, 126]. These capabilities are the core functions that enable the use of a quantum device as either an end node or an internal node of an entanglement-based network. Some of these quantum devices are specialized for use as quantum repeater nodes [85, 92, 120, 121], while others are targeted towards use as end nodes or as multipurpose internal nodes, such as memory equipped junctions between chains of repeater nodes [91, 123, 124].

Progressing beyond fundamental capabilities, several notable experiments have demonstrated the feasibility of entanglement-based quantum networking. End-to-end entanglement was repeatedly and systematically generated in a three node quantum network on the distance scale of a modular computer network [76]. Extending beyond this scale, the first metropolitan scale quantum network links have been demonstrated [93, 114, 115], with [93] demonstrating heralded entanglement generation between fully independent end nodes. These experimental realizations highlight the critical need for shared infrastructure. The three-node network required a middle node with quantum memory to act as an internal network node, with access to a detection based entanglement generation interface shared via a logical switching mechanism. Similarly, the metropolitan-scale architecture in [93] included a midpoint housing both a detection based entanglement generation interface (a BSA) and feedback control mechanisms for aligning quantum state characteristics, including frequency conversion of quantum signals to minimize transmission loss [127] and active phase stabilization protecting against optical phase misalignment which would degrade the entanglement fidelity [128]. In the future, a systems design for a metropolitan hub could extend the midpoint realized in this experiment, with the addition of optical switches allowing the entanglement generation interface and feedback control mechanisms to be shared amongst multiple end nodes.

These examples of shared infrastructure–memory equipped internal nodes, entanglement generation interfaces, switching mechanisms, and feedback control systems–will only increase in importance as networks scale. Repeater chains, for bridging long distances will constitute an additional core infrastructural component. Consequently, architectures and protocols for long distance entanglement-based quantum networks must include these components and provide effective mechanisms for access sharing.

By definition, all entanglement-based quantum networking demonstrations have included an implementation of the physical layer of a quantum network protocol stack. Entanglement generation between independent nodes requires precise alignment between the characteristics of quantum states on each device and tight alignment in the timing of control instructions [25, 129]. In scaling up quantum networks, system architectures must include implementations of the higher layers of the network protocol stack, to ensure applications can be reliably executed by non-neighboring nodes. Transitioning towards the deployment of a full-stack quantum network, experiments have validated two key network layers above the physical layer. In [130], a link layer based on the proposal in [113] was shown to deliver a robust point-to-point entanglement generation service [130] in a two-node network on the distance scale of a modular computer. At the application layer, the development and demonstration of QNodeOS [94] – the first operating system for quantum network nodes – transforms nodes into fully programmable systems. To enable the transition from research systems to deployable long-distance entanglement-based networks, comprehensive systems integration tests are needed to implement and validate all layers of the network protocol stack. These tests will identify outstanding challenges that should guide the active development and optimization of protocol layers and their interfaces.

1.9. A Prototype Full-Stack Quantum Network

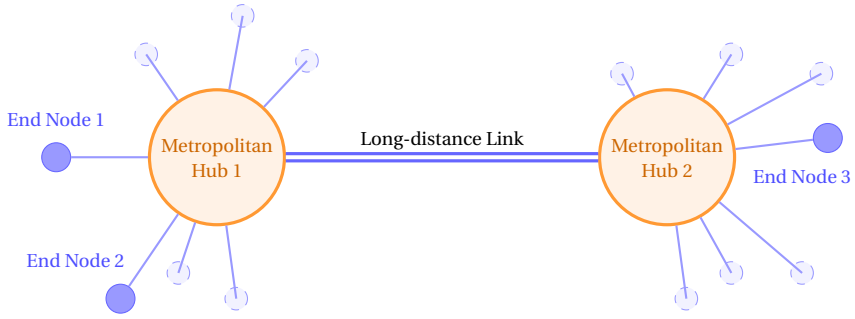


Figure 1.5: Blueprint of the QIA prototype network.

The Quantum Internet Alliance (QIA) has a moonshot mission to deliver a prototype full-stack, long-distance, entanglement-based network before the end of the decade (2029). QIA is a multi-disciplinary collaboration with more the 40 partner institutions from academia, industry, start-ups, and government research organizations. This moonshot mission has a duration of eight years, the first four of which (2022-2026) coincided with this PhD thesis and make up the first phase of the ambitious and technically challenging project.

The prototype network will have the topology indicated in Figure 1.5, which represents two different metropolitan areas linked together by a long-distance backbone of $\geq$ 500km. In each metropolitan area there will be end nodes connected via optical fiber links to a metropolitan hub employing an optical switch to share access to an entanglement generation interface. End nodes may be either simple quantum devices called photonic

clients [131, 132], capable only of either measuring or sending a photonic state, or full quantum processing nodes. The prototype network will include at least two full quantum processing nodes and one photonic client in one metropolitan area, and at least one full quantum processing node in the second metropolitan area.

The full-stack nature of this prototype long-distance, entanglement-based network will be validated in two kinds of systems tests:

1. A two qubit Blind Quantum Computation (BQC) [8–10], between a photonic client and a processing node connected via a metropolitan hub and located in a single metropolitan area. BQC is an example of secure quantum computation in the cloud.

2. Deterministic teleportation [43, 133, 134] of a quantum state between a processing node in the first metropolitan area and a processing node in the second metropolitan area. This application is an example of communicating a quantum state from one end node to another, without converting the state to a classical representation.

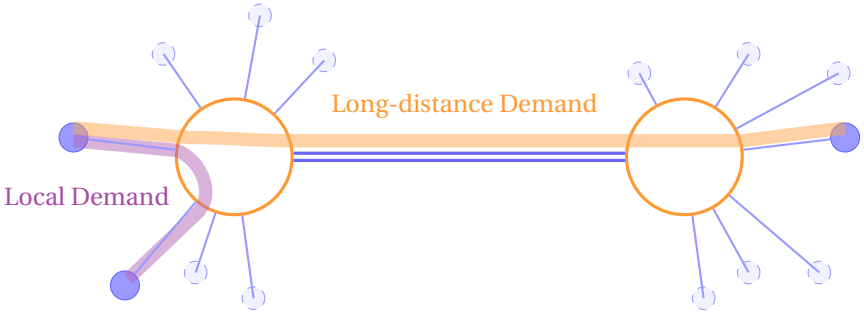


Figure 1.6: Types of demands in a quantum network: local demands between end nodes within the same metropolitan area, and long-distance demands spanning multiple metropolitan hubs.

To enable the BQC systems test, end nodes will demand the generation of end-to-end entangled links requiring the participation of only internal nodes (i.e. the metropolitan hub) within a single metropolitan area. These demands may be referred to as local demands. For the deterministic teleportation systems test, end nodes will submit long-distance demands for end-to-end entanglement requiring use of the long-distance link as well as the participation of internal nodes within both metropolitan areas. Figure 1.6 illustrates two routes through the prototype network associated with an example local demand and an example long-distance demand, respectively.

1.10. PROBLEM STATEMENT

The goal of this research is to enable, for the first time, multi-user access to entanglement-based quantum networks. Realizing these practical quantum networks requires control architectures that operate effectively at multiple scales and manage competing demands from multiple users for shared and limited resources. This thesis addresses three interconnected challenges:

1. **Metropolitan hub control architecture:** Identify the hardware requirements of metropolitan hubs and design a scalable control architecture for metropolitan hubs that can accommodate at least 100 end nodes simultaneously.

2. **Long-distance network control architecture:** Design a control architecture for a long-distance entanglement-based quantum network compatible with the QIA prototype network blueprint (Figure 1.5). This architecture must account for the constraint that end-to-end entanglement generation times exceed the storage times of quantum memories in the prototype network.

3. **Deployment ready implementation:** Implement and validate the control architecture for the prototype network, with particular emphasis on resolving resource contention that arises when local metropolitan area demands and long-distance demands for entanglement compete for the same network resources (Figure 1.6). To support the successful execution of the QIA systems tests, the implementation must prioritize satisfying existing demands for entanglement over serving newly arriving demands.

1.11. Thesis Overview

The contents of the rest of this thesis are as follows:

- In Chapter 2 we develop a control architecture for a metropolitan hub with control of a pool of entanglement generation interfaces. These can be allocated to serve entanglement generation demands originating from a multitude of homogeneous end nodes. As a part of this architecture, we develop a rate control protocol for regulating the rates of entanglement generation demanded by pairs of end nodes.

- In Chapter 3 we propose and validate an alternative control architecture for a scalable metropolitan hub. In this architecture, demands from pairs of end nodes for allocation of an entanglement generation interface are immediately accepted if there is one available, and otherwise the demand is blocked and discarded. In this work, we are able to relax restrictive physical assumptions and allow for the metropolitan hub to serve heterogeneous end nodes. Amongst all possible schemes, the on-demand resource allocation scheme we analyze minimizes the response time with which service decisions are communicated to end nodes.

- In Chapter 4 we propose a centrally controlled architecture for a long-distance entanglement-based quantum network. We introduce a formalism for entanglement generation demands incorporating several key application requirements, including deadlines and minimum fidelity thresholds. Given the constraints imposed by entanglement generation rates and quantum memory storage times, this architecture allocates all internal nodes and links along a demand's end-to-end route for periods of uninterrupted time. The duration of time allocated is calculated based on the rate at which end-to-end links can be successfully generated along the demand's route. To ensure compatibility with the state-of-the-art end-node operating system QNodeOS, our architecture pre-computes and distributes a schedule indicating when each demand will be served. This allows end nodes to

incorporate the network schedule when determining their local order of operations (local schedules) for executing applications.

- In Chapter 5 we propose a suite of control applications that integrate with the centrally controlled architecture developed in Chapter 4 to create an implementation suitable for deployment on the QIA prototype network. We refer to this integrated system as Arqon. In this implementation, the central network controller establishes service agreements with pairs of end nodes whenever a demand is accepted. These service agreements quantitatively specify the quality of service that each demand will receive.

  We define a custom network scheduling application and prove that it deterministically upholds service agreements, preventing newly arriving demands from disrupting those already accepted for service. Through complexity analysis and numeric simulations, we validate that across all investigated parameter regimes, our implementation computes network schedules before the distribution deadline. The timely distribution of schedules facilitates their arrival at all nodes before their start times, though it does not guarantee this outcome.

- In Chapter 6 we conclude by further developing the ties of the control architectures proposed in the preceding chapters to real world implementations of quantum networks. First, we discuss hardware requirements for implementing metropolitan hubs. Then, we discuss experiments planned for January 2026 which will enable a full-stack demonstration of a two node quantum network on the distance scale of a modular quantum computer. These experiments will validate an integration of Arqon with a central controller and the end node operating system QNodeOS.

- Finally, Chapter 7 presents my outlook on the developments required to further drive the technological advancement of quantum networks over the next 5+ years.

REFERENCES

[1] H. Skolimowski, *The structure of thinking in technology,* Technology and Culture **7**, 371 (1966).

[2] W. B. Arthur, *The Nature of Technology: What It Is and How It Evolves* (Allen Lane, an imprint of Penguin Books, 2009).

[3] 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.

[4] A. K. Ekert, *Quantum cryptography based on bell's theorem,* Phys. Rev. Lett. **67**, 661 (1991).

[5] H. Barnum, C. Crepeau, D. Gottesman, A. Smith, and A. Tapp, *Authentication of quantum messages,* in *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, SFCS-02 (IEEE Comput. Soc) p. 449–458.

[6] Y. Kanamori, S.-M. Yoo, D. Gregory, and F. Sheldon, *On quantum authentication protocols,* in *GLOBECOM '05. IEEE Global Telecommunications Conference, 2005.*, Vol. 3 (2005) pp. 5 pp.–.

[7] D. Leichtle, L. Music, E. Kashefi, and H. Ollivier, *Verifying BQP computations on noisy devices with minimal overhead,* PRX Quantum **2**, 040302 (2021).

[8] P. Arright and L. Salvail, *Blind quantum computation,* International Journal of Quantum Information **04**, 883 (2006).

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

[10] A. M. Childs, *Secure assisted quantum computation,* Quantum Information and Computation **5** (2005), 10.26421/QIC5.6.

[11] A. Kent, W. J. Munro, and T. P. Spiller, *Quantum tagging: Authenticating location via quantum information and relativistic signaling constraints,* Physical Review A **84** (2011), 10.1103/physreva.84.012326.

[12] D. Unruh, *Quantum position verification in the random oracle model,* in *Advances in Cryptology – CRYPTO 2014*, edited by J. A. Garay and R. Gennaro (Springer Berlin Heidelberg, Berlin, Heidelberg, 2014) pp. 1–18.

[13] M. Hasanpour, S. Shariat, P. Barnaghi, S. A. Hoseinitabatabaei, S. Vahid, and R. Tafazolli, *Quantum load balancing in ad hoc networks,* Quantum Information Processing **16**, 148 (2017).

[14] D. Ding and L. Jiang, *Coordinating Decisions via Quantum Telepathy,* (2024).

[15] R. Cleve and H. Buhrman, *Substituting quantum entanglement for communication,* Physical Review A **56**, 1201–1204 (1997).

[16] S. Tani, H. Kobayashi, and K. Matsumoto, *Exact Quantum Algorithms for the Leader Election Problem,* ACM Transactions on Computation Theory **4**, 1 (2012).

[17] M. Ben-Or and A. Hassidim, *Fast quantum byzantine agreement,* in *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05 (Association for Computing Machinery, New York, NY, USA, 2005) p. 481–485.

[18] P. Kómár, E. M. Kessler, M. Bishof, L. Jiang, A. S. Sørensen, J. Ye, and M. D. Lukin, *A quantum network of clocks,* Nature Physics **10**, 582–587 (2014).

[19] D. Gottesman, T. Jennewein, and S. Croke, *Longer-baseline telescopes using quantum repeaters,* Phys. Rev. Lett. **109**, 070503 (2012).

[20] A. Broadbent and R. Islam, *Quantum encryption with certified deletion,* in *Theory of Cryptography* (Springer International Publishing, 2020) p. 92–122.

[21] S. Wiesner, *Conjugate coding,* SIGACT News **15**, 78–88 (1983).

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

[23] L. A. Barroso, J. Clidaras, and U. Hölzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, 2nd ed., edited by M. D. Hill, Synthesis Lectures on Computer Architecture (Morgan and Claypool Publishers, 2013).

[24] T. Sterling, M. Anderson, and M. Brodowicz, *High Performance Computing: Modern Systems and Practices*, 1st ed., Computer Architecture and Design (Morgan Kaufmann, Boston, 2017) p. 832.

[25] H. J. Kimble, *The quantum internet,* Nature **453**, 1023–1030 (2008).

[26] S. Wehner, D. Elkouss, and R. Hanson, *Quantum internet: A vision for the road ahead,* Science (2018), 10.1126/science.aam9288.

[27] J. I. Cirac, A. K. Ekert, S. F. Huelga, and C. Macchiavello, *Distributed quantum computation over noisy channels,* Physical Review A **59**, 4249–4254 (1999).

[28] IBM Corporation, *Ibm quantum roadmap 2024: Strategic milestones,* Web page (2024).

[29] Google Quantum AI, *Roadmap: Our quantum computing journey,* Web page (2024).

[30] IonQ, *Ionq forte enterprise: Data center-ready quantum computer,* Web page (2024).

[31] F. Miller, *Telegraphic Code to Insure Privacy and Secrecy in the Transmission of Telegrams* (C.M. Cornwell, 1882).

[32] S. Lloyd, *Universal Quantum Simulators,* Science **273**, 1073 (1996).

[33] S. Aaronson and D. Gottesman, *Improved simulation of stabilizer circuits,* Physical Review A **70** (2004), 10.1103/physreva.70.052328.

[34] S. Bravyi and D. Gosset, *Improved classical simulation of quantum circuits dominated by clifford gates,* Physical Review Letters **116** (2016), 10.1103/physrevlett.116.250501.

[35] C. E. Shannon, *A mathematical theory of communication,* The Bell System Technical Journal **27**, 379 (1948).

[36] F. Bacon, *Of the Advancement and Proficiencie of Learning, or The Partitions of Sciences, IX Bookes* (Leon: Lichfield, Oxford, 1640) Chap. VI.1, pp. 265–266.

[37] B. Schumacher, *Quantum coding,* Phys. Rev. A **51**, 2738 (1995).

[38] J. von Neumann, *Mathematical Foundations of Quantum Mechanics* (Princeton University Press, Princeton, NJ, 1955) p. 445.

[39] P. A. M. Dirac, *The fundamental equations of quantum mechanics,* Proceedings of the Royal Society of London, Series A **109**, 642 (1925).

[40] E. Schrödinger, *Quantisierung als eigenwertproblem (erste mitteilung)*, Annalen der Physik **79**, 361 (1926).

[41] M. Born, *Zur quantenmechanik der stoßvorgänge*, Zeitschrift für Physik **37**, 863 (1926).

[42] IEEE Computer Society, *IEEE Standard for Floating-Point Arithmetic*, IEEE Standard 754-2019 (Institute of Electrical and Electronics Engineers, 2019).

[43] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters, *Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels*, Physical review letters **70**, 1895 (1993).

[44] D. Bouwmeester, J.-W. Pan, K. Mattle, M. Eibl, H. Weinfurter, and A. Zeilinger, *Experimental quantum teleportation*, Nature **390**, 575 (1997).

[45] A. Einstein, B. Podolsky, and N. Rosen, *Can quantum-mechanical description of physical reality be considered complete?* Physical Review **47**, 777 (1935).

[46] E. Schrödinger, *Discussion of probability relations between separated systems*, Proceedings of the Cambridge Philosophical Society **31**, 555 (1935).

[47] J. S. Bell, *On the einstein podolsky rosen paradox*, Physics Physique Fizika **1**, 195 (1964).

[48] A. Aspect, J. Dalibard, and G. Roger, *Experimental test of bell's inequalities using time-varying analyzers*, Physical Review Letters **49**, 1804 (1982).

[49] B. Hensen, H. Bernien, A. E. Dréau, A. Reiserer, N. Kalb, M. S. Blok, J. Ruitenberg, R. F. L. Vermeulen, R. N. Schouten, C. Abellán, W. Amaya, V. Pruneri, M. W. Mitchell, M. Markham, D. J. Twitchen, D. Elkouss, S. Wehner, T. H. Taminiau, and R. Hanson, *Loophole-free Bell inequality violation using electron spins separated by 1.3 kilometres*, Nature **526**, 682 (2015).

[50] R. Jozsa, *Fidelity for mixed quantum states*, Journal of Modern Optics **41**, 2315 (1994).

[51] M. Horodecki, P. Horodecki, and R. Horodecki, *General teleportation channel, singlet fraction, and quasidistillation*, Physical Review A **60**, 1888 (1999).

[52] W. K. Wootters and W. H. Zurek, *A single quantum cannot be cloned*, Nature **299**, 802 (1982).

[53] D. Dieks, *Communication by EPR devices*, Physics Letters A **92**, 271 (1982).

[54] W. Heisenberg, *Über den anschaulichen Inhalt der quantentheoretischen Kinematik und Mechanik*, Zeitschrift für Physik **43**, 172 (1927).

[55] B. van der Vecht, A. T. Yücel, H. Jirovská, and S. Wehner, *Qoala: an application execution environment for quantum internet nodes*, (2025), 2502.17296 [quant-ph] .

[56] D. Deutsch, *Quantum computational networks,* Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences **425**, 73 (1989).

[57] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, *Elementary gates for quantum computation,* Physical Review A **52**, 3457–3467 (1995).

[58] J. H. Plantenberg, P. C. de Groot, C. J. P. M. Harmans, and J. E. Mooij, *Demonstration of controlled-NOT quantum gates on a pair of superconducting quantum bits,* Nature **447**, 836 (2007).

[59] L. Childress, M. V. G. Dutt, J. M. Taylor, A. S. Zibrov, F. Jelezko, J. Wrachtrup, P. R. Hemmer, and M. D. Lukin, *Coherent dynamics of coupled electron and nuclear spin qubits in diamond,* Science **314**, 281 (2006).

[60] M. V. G. Dutt, L. Childress, L. Jiang, E. Togan, J. Maze, F. Jelezko, A. S. Zibrov, P. R. Hemmer, and M. D. Lukin, *Quantum register based on individual electronic and nuclear spin qubits in diamond,* Science **316**, 1312 (2007).

[61] A. Sørensen and K. Mølmer, *Quantum computation with ions in thermal motion,* Physical Review Letters **82**, 1971 (1999).

[62] N. Sangouard, C. Simon, H. de Riedmatten, and N. Gisin, *Quantum repeaters based on atomic ensembles and linear optics,* Rev. Mod. Phys. **83**, 33 (2011).

[63] H. K. Beukers, M. Pasini, H. Choi, D. Englund, R. Hanson, and J. Borregaard, *Remote-entanglement protocols for stationary qubits with photonic interfaces,* PRX Quantum **5** (2024), 10.1103/prxquantum.5.010202.

[64] C. Jones, D. Kim, M. Rakher, P. Kwiat, and T. Ladd, *Design and analysis of communication protocols for quantum repeater networks,* New Journal of Physics **18**, 083015 (2016).

[65] K. Azuma, K. Tamaki, and H.-K. Lo, *All-photonic quantum repeaters,* Nature Communications **6**, 6787 (2015).

[66] S. Muralidharan, L. Li, J. Kim, N. Lütkenhaus, M. D. Lukin, and L. Jiang, *Optimal architectures for long distance quantum communication,* Scientific Reports **6**, 20463 (2016).

[67] P. Walther and A. Zeilinger, *Experimental realization of a photonic Bell-state analyzer,* Physical Review A **72**, 010302 (2005).

[68] M. Michler, K. Mattle, H. Weinfurter, and A. Zeilinger, *Interferometric Bell-state analysis,* Physical Review A **53**, R1209 (1996).

[69] S. L. Braunstein and A. Mann, *Measurement of the Bell operator and quantum teleportation,* Physical Review A **51**, R1727 (1995).

[70] D. C. Burnham and D. L. Weinberg, *Observation of simultaneity in parametric production of optical photon pairs,* Phys. Rev. Lett. **25**, 84 (1970).

[71] P. G. Kwiat, K. Mattle, H. Weinfurter, A. Zeilinger, A. V. Sergienko, and Y. Shih, *New high-intensity source of polarization-entangled photon pairs,* Phys. Rev. Lett. **75**, 4337 (1995).

[72] C. Cabrillo, J. I. Cirac, P. García-Fernández, and P. Zoller, *Creation of entangled states of distant atoms by interference,* Phys. Rev. A **59**, 1025 (1999).

[73] L.-M. Duan, M. D. Lukin, J. I. Cirac, and P. Zoller, *Long-distance quantum communication with atomic ensembles and linear optics,* Nature **414**, 413 (2001).

[74] W. J. Munro, K. Azuma, K. Tamaki, and K. Nemoto, *Inside quantum repeaters,* IEEE Journal of Selected Topics in Quantum Electronics **21**, 78 (2015).

[75] H. J. Briegel, W. Dür, J. I. Cirac, and P. Zoller, *Quantum repeaters for communication,* (1998), arXiv:quant-ph/9803056 [quant-ph] .

[76] 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 **372**, 259 (2021).

[77] J. Calsamiglia and N. Lütkenhaus, *Maximum efficiency of a linear-optical bell-state analyzer,* Applied Physics B **72**, 67–71 (2001).

[78] W. H. Zurek, *Environment-induced superselection rules,* Phys. Rev. D **26**, 1862 (1982).

[79] W. G. Unruh, *Maintaining coherence in quantum computers,* Phys. Rev. A **51**, 992 (1995).

[80] C. Liu, Z. Dutton, C. H. Behroozi, and L. V. Hau, *Observation of coherent optical information storage in an atomic medium using halted light pulses,* Nature **409**, 490 (2001).

[81] D. F. Phillips, A. Fleischhauer, A. Mair, R. L. Walsworth, and M. D. Lukin, *Storage of light in atomic vapor,* Physical Review Letters **86**, 783 (2001).

[82] R. Zhao, Y. O. Dudin, S. D. Jenkins, C. J. Campbell, D. N. Matsukevich, T. A. B. Kennedy, and A. Kuzmich, *Long-lived quantum memory,* Nature Physics **5**, 100 (2009).

[83] M. K. Alqedra, S. P. Horvath, A. Kinos, A. Walther, S. Kröll, and L. Rippe, *Stark control of solid-state quantum memory with spin-wave storage,* Phys. Rev. A **109**, 012607 (2024).

[84] S. P. Horvath, M. K. Alqedra, A. Kinos, A. Walther, J. M. Dahlström, S. Kröll, and L. Rippe, *Noise-free on-demand atomic frequency comb quantum memory,* Phys. Rev. Res. **3**, 023099 (2021).

[85] J. Hänni, A. E. Rodríguez-Moldes, F. Appas, S. Wengerowsky, D. Lago-Rivera, M. Teller, S. Grandi, and H. de Riedmatten, *Heralded entanglement of on-demand spin-wave solid-state quantum memories for multiplexed quantum network links,* Phys. Rev. X **15**, 041003 (2025).

[86] A. Reiserer, N. Kalb, M. S. Blok, K. J. M. van Bemmelen, T. H. Taminiau, R. Hanson, D. J. Twitchen, and M. Markham, *Robust quantum-network memory using decoherence-protected subspaces of nuclear spins,* Phys. Rev. X **6**, 021040 (2016).

[87] I. S. G. Q. K. Distribution, *Quantum Key Distribution (QKD); Control Interface for Software Defined Networks*, Standard (European Telecommunications Standards Institute, Sophia Antipolis, CH, 2022).

[88] V. Martin, J. P. Brito, L. Ortíz, A. Peralta, D. Pitalua-García, A. Santos, A. Fernández, D. Ramos, M. López, C. Duque, A. Aguado, V. López, I. Pau, I. Castro, A. Ruiz-Alba, A. Pastor, J. de la Cruz, S. Domínguez-Boada, and J. Folgueira, *MadQCI: a heterogeneous and scalable SDN-QKD network deployed in production facilities,* npj Quantum Information **10**, 80 (2024).

[89] Y.-A. Chen, Q. Zhang, T.-Y. Chen, W.-Q. Cai, S.-K. Liao, J. Zhang, K. Chen, J. Yin, J.-G. Ren, Z. Chen, S.-L. Han, Q. Yu, K. Liang, F. Zhou, X. Yuan, M.-S. Zhao, T.-Y. Wang, X. Jiang, L. Zhang, W.-Y. Liu, Y. Li, Q. Shen, Y. Cao, C.-Y. Lu, R. Shu, J.-Y. Wang, L. Li, N.-L. Liu, F. Xu, X.-B. Wang, C.-Z. Peng, and J.-W. Pan, *An integrated space-to-ground quantum communication network over 4,600 kilometres,* Nature **589**, 214 (2021).

[90] M. Sasaki, M. Fujiwara, H. Ishizuka, W. Klaus, K. Wakui, M. Takeoka, S. Miki, T. Yamashita, Z. Wang, A. Tanaka, K. Yoshino, Y. Nambu, S. Takahashi, A. Tajima, A. Tomita, T. Domeki, T. Hasegawa, Y. Sakai, H. Kobayashi, T. Asai, K. Shimizu, T. Tokura, T. Tsurumaru, M. Matsui, T. Honjo, K. Tamaki, H. Takesue, Y. Tokura, J. F. Dynes, A. R. Dixon, A. W. Sharpe, Z. L. Yuan, A. J. Shields, S. Uchikoga, M. Legré, S. Robyr, P. Trinkler, L. Monat, J.-B. Page, G. Ribordy, A. Poppe, A. Allacher, O. Maurhart, T. Länger, M. Peev, and A. Zeilinger, *Field test of quantum key distribution in the Tokyo QKD Network,* Optics Express **19**, 10387 (2011).

[91] 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. **130**, 050803 (2023).

[92] V. Krutyanskiy, M. Canteri, M. Meraner, J. Bate, V. Krcmarsky, J. Schupp, N. Sangouard, and B. P. Lanyon, *Telecom-wavelength quantum repeater node based on a trapped-ion processor,* Phys. Rev. Lett. **130**, 213601 (2023).

[93] A. J. Stolk, K. L. van der Enden, M.-C. Slater, I. te Raa-Derckx, P. Botma, J. van Rantwijk, J. J. B. Biemond, R. A. J. Hagen, R. W. Herfst, W. D. Koek, A. J. H. Meskers, R. Vollmer, E. J. van Zwet, M. Markham, A. M. Edmonds, J. F. Geus, F. Elsen, B. Jungbluth, C. Haefner, C. Tresp, J. Stuhler, S. Ritter, and R. Hanson, *Metropolitan-scale heralded entanglement of solid-state qubits,* Science Advances **10**, eadp6442 (2024).

[94] C. Delle Donne, M. Iuliano, B. van der Vecht, G. M. Ferreira, H. Jirovská, T. J. W. van der Steenhoven, A. Dahlberg, M. Skrzypczyk, D. Fioretto, M. Teller, P. Filippov, A. R.-P. Montblanch, J. Fischer, H. B. van Ommen, N. Demetriou, D. Leichtle, L. Music, H. Ollivier, I. te Raa, W. Kozlowski, T. H. Taminiau, P. Pawełczak, T. E. Northup, R. Hanson, and S. Wehner, *An operating system for executing applications on quantum network nodes,* Nature **639**, 321–328 (2025).

[95] D. Ferrari, S. Carretta, and M. Amoretti, *A modular quantum compilation framework for distributed quantum computing,* IEEE Transactions on Quantum Engineering **4**, 1 (2023).

[96] C. Monroe, R. Raussendorf, A. Ruthven, K. R. Brown, P. Maunz, L.-M. Duan, and J. Kim, *Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects,* Phys. Rev. A **89**, 022317 (2014).

[97] H. Shapourian, E. Kaur, T. Sewell, J. Zhao, M. Kilzer, R. Kompella, and R. Nejabati, *Quantum data center infrastructures: A scalable architectural design perspective,* (2025), arXiv:2501.05598 [quant-ph] .

[98] J. C. Maxwell, *Viii. a dynamical theory of the electromagnetic field,* Philosophical Transactions of the Royal Society of London **155**, 459 (1865), https://royalsocietypublishing.org/doi/pdf/10.1098/rstl.1865.0008 .

[99] A. Einstein, *Zur Elektrodynamik bewegter Körper,* Annalen der Physik **17**, 891 (1905).

[100] L. L. Peterson and B. S. Davie, *Computer Networks: A Systems Approach*, 6th ed. (Morgan Kaufmann, 2021).

[101] Y. Zheng, C. Zhai, D. Liu, J. Mao, X. Chen, T. Dai, J. Huang, J. Bao, Z. Fu, Y. Tong, X. Zhou, Y. Yang, B. Tang, Z. Li, Y. Li, Q. Gong, H. K. Tsang, D. Dai, and J. Wang, *Multichip multidimensional quantum networks with entanglement retrievability,* Science **381**, 221 (2023).

[102] D. Llewellyn, Y. Ding, I. I. Faruque, S. Paesani, D. Bacco, R. Santagati, Y. J. Qian, Y. Li, Y. F. Xiao, M. Huber, M. Malik, G. F. Sinclair, X. Zhou, K. Rottwitt, J. L. O'Brien, J. G. Rarity, Q. Gong, L. K. Oxenlowe, J. Wang, and M. G. Thompson, *Chip-to-chip quantum teleportation and multi-photon entanglement in silicon,* Nature Physics **16**, 148 (2020).

[103] M. Akhtar, F. Bonus, F. R. Lebrun-Gallagher, N. I. Husain, S. Murali, S. Martina, L. Schuhmann, F. Schmidt-Kaler, S. Weidt, and W. K. Hensinger, *A high-fidelity quantum matter-link between ion-trap microchip modules,* Nature Communications **14** (2023), 10.1038/s41467-022-35285-3.

[104] D. Main, P. Drmota, D. P. Nadlinger, E. M. Ainley, G. Araneda, R. Srinivas, M. H. Abobeih, C. J. Ballance, and D. M. Lucas, *Distributed quantum computing across an optical network link,* Nature **638**, 383 (2025).

[105] V. Cerf, Y. Dalal, and C. Sunshine, *Specification of Internet Transmission Control Program*, RFC 675 (Network Working Group, 1974).

[106] World Population Review, *Internet users by country 2025,* (2025).

[107] UNCTAD, *Business e-commerce sales and the role of online platforms*, UNCTAD Technical notes on ICT for development No. 1 (United Nations Conference on Trade and Development, Geneva, 2024).

[108] J. C. Doyle, D. L. Alderson, L. Li, S. Low, M. Roughan, S. Shalunov, R. Tanaka, and W. Willinger, *The "robust yet fragile" nature of the internet,* Proceedings of the National Academy of Sciences **102**, 14497 (2005), https://www.pnas.org/doi/pdf/10.1073/pnas.0501426102 .

[109] J. Meza, T. Xu, K. Veeraraghavan, and O. Mutlu, *A large scale study of data center network reliability,* in *Proceedings of the Internet Measurement Conference 2018*, IMC '18 (Association for Computing Machinery, New York, NY, USA, 2018) p. 393–407.

[110] R. P. Jindal, *From millibits to terabits per second and beyond - over 60 years of innovation,* in *2009 2nd International Workshop on Electron Devices and Semiconductor Technology* (2009) pp. 1–6.

[111] A. Sharma and A. H. Lashkari, *A survey on encrypted network traffic: A comprehensive survey of identification/classification techniques, challenges, and future directions,* Comput. Netw. **257** (2025), 10.1016/j.comnet.2024.110984.

[112] H. Zimmermann, *Osi reference model - the iso model of architecture for open systems interconnection,* IEEE Transactions on Communications **28**, 425 (1980).

[113] 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* (2019) pp. 159–173.

[114] C. M. Knaut, A. Suleymanzade, Y.-C. Wei, D. R. Assumpcao, P.-J. Stas, Y. Q. Huan, B. Machielse, E. N. Knall, M. Sutula, G. Baranes, N. Sinclair, C. De-Eknamkul, D. S. Levonian, M. K. Bhaskar, H. Park, M. Lončar, and M. D. Lukin, *Entanglement of nanophotonic quantum memory nodes in a telecom network,* Nature **629**, 573–578 (2024).

[115] J. L. Liu, X. Y. Luo, Y. Yu, C.-Y. Wang, B. Wang, Y. Hu, J. Li, M.-Y. Zheng, B. Yao, Z. Yan, D. Teng, J.-W. Jiang, X.-B. Liu, X.-P. Xie, J. Zhang, Q.-H. Mao, X. Jiang, Q. Zhang, X.-H. Bao, and J.-W. Pan, *Creation of memory–memory entanglement in a metropolitan quantum network,* Nature **629**, 579–585 (2024).

[116] 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 **5**, 186 (2022).

[117] 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 **3**, 538 (2007).

[118] 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 **558**, 268 (2018).

[119] 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 **497**, 86 (2013).

[120] 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 **438**, 828 (2005).

[121] 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 **316**, 1316 (2007).

[122] 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 **607**, 69 (2022).

[123] C. E. Bradley, J. Randall, M. H. Abobeih, R. C. Berrevoets, M. J. Degen, M. A. Bakker, M. Markham, D. J. Twitchen, and T. H. Taminiau, *A ten-qubit solid-state spin register with quantum memory up to one minute,* Phys. Rev. X **9**, 031045 (2019).

[124] C. J. Ballance, T. P. Harty, N. M. Linke, M. A. Sepiol, and D. M. Lucas, *High-fidelity quantum logic gates using trapped-ion hyperfine qubits,* Phys. Rev. Lett. **117**, 060504 (2016).

[125] T. H. Taminiau, J. Cramer, T. van der Sar, V. V. Dobrovitski, and R. Hanson, *Universal control and error correction in multi-qubit spin registers in diamond,* Nature Nanotechnology **9**, 171–176 (2014).

[126] P. Drmota, D. Main, D. P. Nadlinger, B. C. Nichol, M. A. Weber, E. M. Ainley, A. Agrawal, R. Srinivas, G. Araneda, C. J. Ballance, and D. M. Lucas, *Robust quantum memory in a trapped-ion quantum network node,* Phys. Rev. Lett. **130**, 090803 (2023).

[127] A. Stolk, K. van der Enden, M.-C. Roehsner, A. Teepe, S. Faes, C. Bradley, S. Cadot, J. van Rantwijk, I. te Raa, R. Hagen, A. Verlaan, J. Biemond, A. Khorev, R. Vollmer, M. Markham, A. Edmonds, J. Morits, T. Taminiau, E. van Zwet, and R. Hanson, *Telecom-band quantum interference of frequency-converted photons from remote detuned nv centers,* PRX Quantum **3** (2022), 10.1103/prxquantum.3.020359.

[128] A. Stolk, J. Biemond, K. van der Enden, L. van Dooren, E. van Zwet, and R. Hanson, *Extendable optical phase synchronization of remote and independent quantum network nodes over deployed fibers,* Phys. Rev. Appl. **23**, 014077 (2025).

[129] S. D. Barrett and P. Kok, *Efficient high-fidelity quantum computation using matter qubits and linear optics,* Physical Review A **71**, 060310 (2005).

[130] 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 **8**, 121 (2022).

[131] 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 **132** (2024), 10.1103/physrevlett.132.150604.

[132] J. van Dam, G. Avis, T. B. Propp, F. Ferreira da Silva, J. A. Slater, T. E. Northup, and S. Wehner, *Hardware requirements for trapped-ion-based verifiable blind quantum computing with a measurement-only client,* Quantum Science and Technology **9**, 045031 (2024).

[133] M. Riebe, H. Häffner, C. F. Roos, W. Hänsel, J. Benhelm, G. P. T. Lancaster, T. W. Körber, C. Becher, F. Schmidt-Kaler, D. F. V. James, and R. Blatt, *Deterministic quantum teleportation with atoms,* Nature **429**, 734 (2004).

[134] 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 **429**, 737 (2004).

2

AN ARCHITECTURE FOR CONTROL OF ENTANGLEMENT GENERATION SWITCHES IN QUANTUM NETWORKS

Entanglement between quantum network nodes is often produced using intermediary devices - such as heralding stations - as a resource. When scaling quantum networks to many nodes, requiring a dedicated intermediary device for every pair of nodes introduces high costs. Here, we propose a cost-effective architecture to connect many quantum network nodes via a central quantum network hub called an Entanglement Generation Switch (EGS). The EGS allows multiple quantum nodes to be connected at a fixed resource cost, by sharing the resources needed to make entanglement. We propose an algorithm called the Rate Control Protocol (RCP) which moderates the level of competition for access to the hub's resources between sets of users. We proceed to prove a convergence theorem for rates yielded by the algorithm. To derive the algorithm we work in the framework of Network Utility Maximization (NUM) and make use of the theory of Lagrange multipliers and Lagrangian duality. Our EGS architecture lays the groundwork for developing control architectures compatible with other types of quantum network hubs as well as system models of greater complexity.

2.1. INTRODUCTION

A quantum network enables radically new capabilities that are provably impossible to attain in any classical network [1]. Examples include applications such as secure communication [2, 3], secure quantum computing in the cloud [4, 5], and clock synchronization [6]. Users utilize the end nodes of a network to run applications. The key to unlocking widespread roll-out of these applications is the ability to produce entanglement between these end nodes.

Prevalent methods for generating entanglement between two quantum nodes that are directly connected by a quantum communication medium (e.g., optical fibers) involve

an intermediate device. A prime example is heralded entanglement generation [7, 8] in which the intermediary device is a so-called heralding station. This method of producing entanglement has successfully been demonstrated in many experimental platforms including Color Centers [9, 10], Ion Traps [11, 12], Atomic Ensembles [13, 14] and Neutral Atoms [15]. As quantum networks continue to scale, it becomes increasingly impractical to maintain direct fiber connections and dedicated heralding stations for every pair of end nodes.

To address this challenge, we propose a scalable quantum network architecture for an Entanglement Generation Switch (EGS), a central hub equipped with a limited number of intermediate devices called resources, a switch, and a processor responsible for managing a scheduling algorithm and sending classical messages to nodes. This central hub enables multiple nodes to share the intermediate devices, significantly reducing the complexity and total resources required for large-scale deployment. While our results apply to an EGS sharing any type of entanglement generation resource, a specific example illustrates how an EGS can operate: Consider quantum network nodes that generate entanglement between them using the so-called single-click bipartite entanglement generation protocol (see e.g [10]). In this case the resource(s) to be shared are the heralding station(s). Such stations consist of two input channels connected to a 50/50 beam splitter, which is then connected by two output channels to a pair of photon detectors that are each connected to a device for processing the measurement outcomes, such as a Field Programmable Gate Array (FPGA). The basic principle of the single-click protocol requires that each network node of the pair locally generates entanglement between a qubit in their local memory and a travelling photon. The photon is sent to a heralding station at which an entanglement swap is attempted on the two photons received; if the entanglement swap is successful, the qubits of the two network nodes will have become entangled. An EGS aims to share one or more heralding stations amongst many connected network nodes. These nodes will still run the single-click protocol, but be limited to using the heralding station needed in the time allocated to them by the EGS.

A crucial challenge in implementing such an architecture is the efficient allocation of the central hub's resources to different pairs of users in distinct time slots. Similar to classical networking, the allocation process should be driven by user demand for network resources. In the context of quantum networks, this translates to the demand of a user pair (u_i, u_j) for entanglement generation at a specific rate or fidelity. Given a set of user demands, the EGS must compose a schedule for the allocation of resources in order to service those demands. In general, the total demand of users may exceed the available resources at the central hub, leading to scheduling and resource allocation challenges.

Here, we introduce the first algorithm for regulating user demand to an EGS, thereby solving this key challenge. Specifically, the algorithm takes as input a vector of rates of entanglement generation demanded by pairs of users and outputs an updated rate vector. The current set of user-originated demands is a measure of competition for EGS resources. We construct the algorithm within the Network Utility Maximization (NUM) framework, wherein the problem of demand regulation is cast as a constrained optimization problem. To solve the problem, we derive the algorithm by using the theory of Lagrange multipliers and Lagrangian duality. These tools, respectively, enable including the constraints together with the objective of the optimization problem and solving for

a parameter vector which is the unknown value of the combined problem. Regulating competition for the resources by modifying user demand makes it possible to enforce a notion of fairness in the allocation of resources and maximize resource utilization. Since the algorithm regulates competition by calculating the rates demanded by users, we call it the Rate Control Protocol (RCP).

2.1.1. RESULTS SUMMARY

We make the following contributions:

- We characterize (Theorem 1) the capacity region of the EGS, which is the maximal set of rates at which users can demand entanglement generation such that there exists a scheduling policy under which, on average, the demanded rates do not exceed the delivered rates. The impact of specifying the capacity region is that it delineates which rates can feasibly be serviced by the EGS.

- We prove (Theorem 1) that under the Maximum Weight Scheduling policy (Definition 6) for resource allocation it is possible for the EGS to deliver average rates of entanglement generation that match the requested rates, for any rate vector from within the capacity region. Therefore, an EGS operated with this scheduling policy can achieve throughput optimality as long as the rates demanded by users lie withing the capacity region. To prove the theorem, we use the Lyapunov stability theory of Markov chains.

- We derive the RCP, an algorithm to regulate the rates of bi-partite entanglement generation which pairs of users demand from an EGS. The RCP solves the problem of moderating user competition for EGS resources. The derivation is based on techniques from Network Utility Maximization (NUM) and its quantum network extension (QNUM), where resource allocation in a (quantum) network is modelled as an optimization problem that can be solved using methods from convex optimization theory.

- We prove (Theorem 4) that the sequence of arrival rate vectors yielded by the RCP converges over time slots to an optimum value, given any feasible rate vector as initial condition. The significance of this result is that if the RCP is used to set the demand rates of entanglement generation over a series of time-slots, the set of demanded rates will approach an optimal value, as long as the initial rate vector supplied to the algorithm is feasible. The proof relies on Lagrange multipliers and Lagrangian duality theory.

- Finally, we supply numerical results that support our analysis.

2.1.2. RELATED WORK

A quantum network hub that can store locally at least one qubit per linked node and distributes entanglement across these links has been studied [16, 17]. We refer to such a hub as an Entanglement Distribution Switch (EDS). This system differs from our system because the central hub has qubits and/or quantum memories, whereas our system does not. In [16] the focus is on assessing the EDS performance in terms of the rate at which it

creates $n-$partite entanglements, and in [17] the possible rate/fidelity combinations of GHZ states that may be supplied by an EDS [17] are studied.

Maximum Weight scheduling is a type of solution to the problem of resource allocation which is based on assigning resources to sets of users with the largest service backlog. A Maximum Weight scheduling policy was originally presented in [18] for resource allocation in classical communication networks and was adapted to the analysis of a single switch for classical networking in [19], where it was shown that under this scheduling policy the set of request arrival rates matches the request departure rates (or in other words the policy stabilizes the switch for all feasible arrival rates). In [20] the capacity region of an EDS, defined as the set of arrival rates of requests for end-to-end multi-partite entanglements that stabilize the switch, is first characterized. Using the Lyapunov stability theory of Markov chains, a Maximum Weight scheduling policy is proposed and shown to stabilize the switch for all arrival rates within the capacity region. To summarize, in each of the classical network settings and in the EDS setting a Maximum Weight scheduling policy has the merit of achieving a specified performance metric. None of these results are immediately applicable to our system. We demonstrate that such a policy achieves the performance metric of throughput optimality when applied to the EGS by first characterizing the capacity region of the EGS, which has not been done before, and then proving that a Maximum Weight scheduling policy also achieves throughput optimality in our system.

These results on the analysis of EDS systems constitute the first analytic approaches to resource allocation by a quantum network hub. However, due to the assumption that an EDS locally controls some number of qubits per link, the system has a high technical implementation cost which may not be compatible with near-term quantum networks. Moreover, although these works assume that there is competition between multiple sets of users, the focus is purely on the capacity of the EDS system. Conversely, our analytic contributions apply to EGS quantum network hubs, which have a low technical implementation cost because the hub does not require local control of any qubits or quantum memory. Furthermore, our results extend beyond the analysis of the capacity of the EGS and we propose the RCP as a solution to the problem of moderating competition for the EGS resources.

In [21], a quantum network topology is studied where user-controlled nodes are connected through a hub known as a Qonnector. The Qonnector provides the necessary hardware for limited end nodes to execute applications in pairs or small groups. A potential configuration of the Qonnector is as an EGS. While [21] focuses on assessing the performance of certain applications in this topology, it does not address control policies for the system. In contrast, our work examines control policies for an EGS.

NUM was first introduced in [22] and has been widely used to develop and analyze control policies for classical networks [23]. It is a powerful framework for designing and analyzing communication protocols in classical networks wherein the problem of allocating resources amongst competing sets of users is cast as a constrained optimization problem. This framework was recently extended to QNUM by [24]. Therein, the authors first develop three performance metrics and use them to catalogue the utility of resource allocation in a quantum network model where each link is associated with a rate and fidelity of entanglement delivery to communicating users. This work does not imme-

diately extend to control policies, as the resource allocations investigated are based on static numerical optimization and need to be recalculated in response to changes in the constraints or sets of users.

In classical networks, probabilistic failures such as loss of a message during transmission or irreconcilable distortion due to transmission over a noisy channel may occur. A serious challenge introduced in the analysis of quantum networks is that in addition to the failure modes of a classical network several new probabilistic failure modes arise that are independent of the state of the network but nevertheless affect its ability to satisfy demands. An example is the probabilistic success in practical realizations of heralded entanglement generation [9–15]. Due to this failure mode, scheduling access to a resource at a certain rate does not guarantee entanglement generation at that rate, thereby complicating the analysis of scheduling.

It is important to distinguish between the concept of rates in classical network control protocols and the notion of rate in the model of a quantum network hub presented here. In classical networks, users transmit *data* at some rate and classical network control protocols, such as the Transmission Control Protocol (TCP), regulate the rate at which users send their data [23]. In contrast, in our quantum network hub model, users demand a rate of entanglement generation. However, a significant challenge in developing a control protocol for the EGS is the difference between the rate of attempted entanglement generation and the rate at which entanglement delivery is demanded and delivered to users. Explicitly, in the RCP it is the desired rates of entanglement generation that serve as the controllable parameters moderated by the protocol.

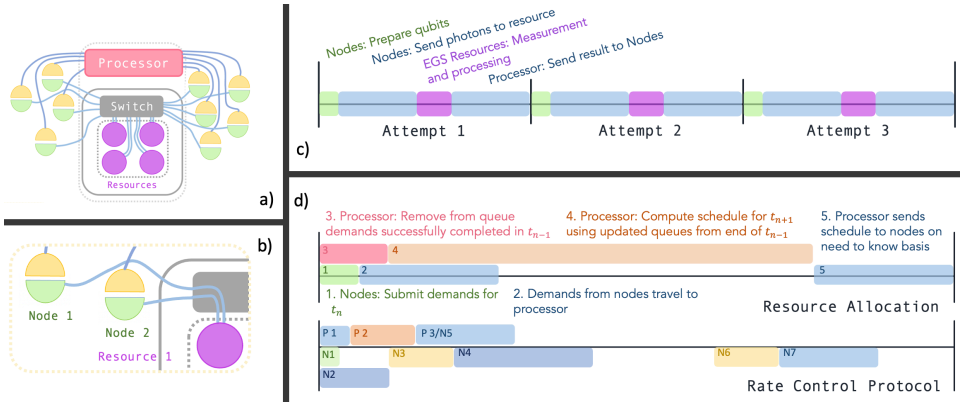


Figure 2.1: EGS Architecture: a) EGS structure: An EGS with $R = 4$ resources connected to $N = 9$ nodes. The EGS is controlled by a classical processor and consists of a switch, resources, and physical connections. Nodes have quantum communication channels to the switch and classical communication channels to the processor. b) Resource Allocation: The switch opens connections to link nodes 1, 2 and resource 1. For example, the connections may consist of direct optical fiber paths from the nodes to the switch and from the switch to the resource, via an interface at the switch. This establishes the physical allocation of resource 1 to the communication session of nodes 1, 2 for time slot t_n. c) Quantum communication sequence: Node-to-processor communication in time slot t_n with a batch size of three entanglement generation attempts. d) Concurrent classical communication sequences: Nodes and the processor communicate in time slot t_n, governing resource allocation and the RCP (see Algorithm 1 for RCP details.)

Table 2.1: Inventory of notation introduced in Section 2.2

Identifier	Description	Domain
U	the set of (user operated) quantum network nodes, of cardinality $\|U\| = N$	$\mathbb{N}$
R	the number of resources controlled by the EGS processor	$\mathbb{N}$
t_n	the $n-th$ time-slot of the EGS system	$\mathbb{N}$
S	the set of communication sessions	$\{1, \cdots, N\}^2$
$\boldsymbol{\lambda}(t_n)$	the vector of target rates of all communication sessions at time t_n	$\mathbb{R}^{+\|S\|}$
$a_s(t_n)$	the number of demands from communication session s in time-slot t_n	$\mathbb{N}$
$\boldsymbol{q}(t_n)$	the vector of queues, with components $\big(q_s(t_n)\ \forall s\big)$	$\mathbb{N}^{\|S\|}$
$\boldsymbol{M}(t_n)$	the resource allocation schedule for time-slot t_n, with components $\big(M_s(t_n)\ \forall s\big)$	$\mathbb{N}^{\|S\|}$
x_s	the maximum number of resources that can be allocated to communication session s in any one time-slot	$\{1, \cdots, R\}$
p_{gen}	the probability a communication session allocated a resource for one time-slot successfully generates entanglement	$[0, 1]$
$g_s(t_n)$	the number of successfully generated entangled pairs by communication session s in t_n	$\{0, 1, \cdots, M_s(t_n)\}$
$\mathcal{C}$	the set which has the capacity region of the EGS as interior	$\mathbb{R}^+$
λ_{EGS}	the maximum total rate that can be delivered, on average, by the EGS, $\lambda_{\text{EGS}} = R \cdot p_{\text{gen}}$	$\mathbb{R}^+$
$\lambda_{\text{gen},s}^{\max}$	the maximum rate the EGS can deliver, on average, to communication session s, $\lambda_{\text{gen},s}^{\max} = x_s \cdot p_{\text{gen}}$	$\mathbb{R}^+$
$\lambda_s^{\min}$	a minimum acceptable rate of entanglement generation specified by communication session s	$\mathbb{R}^+$
λ_u	the maximum rate at which each node $u \in U$ can generate and/or make use of entanglement, across all of the sessions that it is involved in	$\mathbb{R}^+$

2.2. PRELIMINARIES

Operation of the EGS requires interactions between the set of quantum network nodes U and the EGS processor with control over R resources. See Fig. 2.1 a) for an overview of the physical architecture. Below we delineate the process by which pairs of nodes may request (Fig. 2.1 d)) and receive (Fig. 2.1 b) and d)) resource allocations from the processor. We assume:

- the EGS operates in a fixed-duration time slotted system where t_n denotes the n^{th} time slot;
- timing synchronization between the processor and each node is continuously managed by classical control electronics at the physical layer;
- allocation of a single resource to communication session s for one time slot allows for the creation of a maximum of one entangled pair with a success probability of p_{gen}. A consistent physical model involves a *batched sequence* of attempts, which can be terminated upon the successful creation of an entangled pair or at the end of the time slot. See Fig. 2.1 c) for an example quantum communication sequence compatible with heralded entanglement generation.

The classical communication sequence repeated in each time slot t_n which governs resource allocation is summarized in Fig. 2.1 d). In what follows we introduce and explain each step of this communication sequence. The notation introduced throughout this section is summarized in Table 2.1.

2.2.1. DEMANDS FOR RESOURCE ALLOCATION FROM NODES TO THE EGS PROCESSOR

Definition 1 (Target Rate, Communication Session)**.** *Each possible pair of nodes has the potential to require shared bipartite entanglement. To fulfill this need, a node pair (u_i, u_j) requires the processor to allocate a resource. The node pair sets a target rate $\lambda_{(i,j)}(t_n)$ once per time slot, which represents the average number of entangled pairs per time slot they aim to generate using one or more EGS resources. A distinct pair of nodes with a non-zero target rate is referred to as a communication session and is associated with a unique communication session ID, s. The set of communication session IDs, S is defined as follows:*

$$S := \left\{ s = (i,j) \mid i < j \text{ and } \lambda_s(t_n) > 0, \forall\ (i,j) \in \{1, \cdots, N\}^2 \right\} \tag{2.1}$$

where $N = |U|$ is the total number of network nodes with connections to the EGS.

Henceforth each pair of nodes will be identified by its communication session id s. The target rates of all communication sessions in time-slot t_n can be written as a vector $\boldsymbol{\lambda}(t_n) \in \mathbb{R}^{|S|}$, the s^{th} component of which is labelled by communication session ID s as $\lambda_s(t_n)$.

A rate of entanglement generation is the service demanded by each communication session from the EGS. To address the difference between the desired rate and the rate at which a communication session requires resource allocation to achieve that rate, we establish the following model for demand, which is compatible with a discrete time scheduling policy.

Definition 2 (Demand)**.** *Demands for resources are requests made by communication session s to obtain a single entangled pair. The number of demands $a_s(t_n)$ submitted by session s at time slot t_n depends on its target rate $\lambda_s(t_n)$. If $\lambda_s(t_n) > 1$, then communication session s first submits $\lfloor \lambda_s(t_n) \rfloor$ demands. For a communication session s with $0 \leq \lambda_s(t_n) \leq 1$, or to account for the remaining part of the rate for any session with $\lambda_s(t_n) > 1$, each communication session randomly generates demands by sampling from a Bernoulli distribution with a mean equal to $\lambda_s(t_n) - \lfloor \lambda_s(t_n) \rfloor$, so that in general the submitted demands satisfy a (shifted) Bernoulli distribution, $a_s(t_n) \sim \text{Bernoulli}\big(\lambda_s(t_n) - \lfloor \lambda_s(t_n) \rfloor\big) + \lfloor \lambda_s(t_n) \rfloor$.*

Definition 3 (Designated Communication Node, Secondary Node)**.** *One of the nodes of every communication session is marked as the designated communication node for communicating the entanglement requests to the switch. The terms designated communication node and secondary node are used to refer to the two nodes of a communication session.*

2.2.2. PROCESSING DEMANDS FOR RESOURCE ALLOCATION

Definition 4 (Queue)**.** *When the processor receives a demand, it is added to one of $|S|$ queues, one for each communication session. The set of demands received by the processor by time-slot t_n and not yet satisfied is captured by the queue vector $\mathbf{q}(t_n) \in \mathbb{N}^{|S|} = (q_s(t_n)\ \forall s)$, where the component $q_s(t_n)$ is the queue of communication session s at time t_n. Each queue processes demands in first in first out order. As all demands are identical, we interchangeably use $q_s(t_n)$ to refer to both the queue length of communication session s in time slot t_n and the queue itself.*

Definition 5 ((Demand-Based) Schedule)**.** *A resource allocation schedule is a vector*

$$\boldsymbol{M}(t_{n+1}) \in \mathbb{N}^{|S|} \tag{2.2}$$

calculated by the EGS processor in time slot t_n determining the assignment of the resources for time slot t_{n+1}. A single session s may be allocated the use of multiple resources, up to a maximum number x_s set by the EGS which does not exceed R, the total number of resources controlled by the EGS. For every session $s \in S$ the entry

$$M_s(t_{n+1}) \in \{0,\ 1,\ \cdots,\ x_s\} \tag{2.3}$$

corresponds to the number of resources assigned to s for the entire duration of time slot t_{n+1}. A demand based schedule is based on the vector of all queues, $\boldsymbol{q}(t_n)$, as it stands before new demands are registered in t_n, and satisfies

$$\sum_s M_s(t_{n+1}) \leq \min\big(\sum_s q_s(t_n),\ R\big), \tag{2.4}$$

$$0 \leq M_s(t_{n+1}) \leq \min\big(q_s(t_n),\ x_s\big) \leq R,\ \forall\ s. \tag{2.5}$$

Each node of a communication session s requires a physical connection to the EGS switch. A single physical connection, such as an optical fiber, can be used for this purpose. To enable multiple connections between a node and the switch, options include the use of optical multiplexers over a single fiber or utilizing multiple fibers within a fiber bundle. The parameters $\big(x_s\ \forall s\big)$ are motivated by situations where the number of physical connections that can be dedicated to service communication session s are limited.

Definition 6 (Maximum Weight Scheduling)**.** *The set $\mathscr{M}$ of feasible demand based schedules at time slot t_n contains all vectors $\mathbf{M}'(t_{n+1}) \in \mathbb{N}^{|S|}$ satisfying (2.3), (2.4), and (2.5). The EGS processor selects a maximum weight schedule $\mathbf{M}(t_{n+1}) \in \mathscr{M}$ from the feasible schedules for the following time slot by solving for*

$$\mathbf{M}(t_{n+1}) \in arg\ \max_{\mathbf{M}'} \sum_s q_s(t_n) M'_s(t_{n+1}). \tag{2.6}$$

In words, the schedule is selected from the set of feasible schedules by first solving for the subset of schedules that allocate resources to the sessions with the largest number of queued demands. If that subset contains more than one schedule, a schedule is randomly selected from the subset.

By the end of t_n, the schedule for t_{n+1} has been computed by the processor and broadcast to the nodes. If the schedule allocates use of a resource to communication session s for t_{n+1}, the users of s utilize the allocated resource to make a batch of entanglement generation attempts over the duration of t_{n+1}. The demand at the front of queue s is only marked as served once both a resource has been allocated and the users of s have successfully generated entanglement. Hence the dynamics of each queue are given by,

$$q_s(t_{n+1}) = [q_s(t_n) + a_s(t_n) - g_s(t_n)]^+\ \forall\ s, \tag{2.7}$$

where $[z]^+ = \max(z, 0)$, and $g_s(t_n)$ is the number of successfully generated entangled pairs by s during t_n. In words, every subsequent time slot the demands that arrived

in the previous time slot are added to the queue and those that were scheduled and successfully resulted in the generation of an entangled pair are removed from the queue. The updated queue is always of non-negative length since the number of successfully generated entangled pairs is a sample of a binomial random variable where the number of trials is the number of resources allocated to s, $M_s(t_n)\big(\leq q_s(t_n)\big)$, and the trial success probability is p_{gen},

$$g_s(t_n) \sim \text{Bin}\big(M_s(t_n),\ p_{\text{gen}}\big).$$

Definition 7 (Supportable rate)**.** *The arrival rate vector $\boldsymbol{\lambda}(t_n) \in \mathbb{R}^{+|S|} = \big(\lambda_s(t_n)\ \forall\ s\big)^T$ is supportable if there exists a schedule under which,*

$$\lim_{Q\to\infty} \lim_{n\to\infty} P\big(|\boldsymbol{q}(t_n)| \geq Q\big) = 0, \tag{2.8}$$

where $|\boldsymbol{q}(t_n)| := \sum_s |q_s(t_n)|$ is the sum of the number of demands in the queue of each session in time slot t_n. That is, $\boldsymbol{\lambda}(t_n)$ is supportable if the probability that the total queue length becomes infinite is zero.

Definition 8 (Capacity Region)**.** *The capacity region of an EGS is the set of arrival rate vectors that are supportable by the EGS. For each rate vector $\boldsymbol{\lambda}$ in the capacity region, there exists some scheduling routine such that an EGS operating under that scheduling algorithm can support the rate vector $\boldsymbol{\lambda}$.*

If the rate vector $\boldsymbol{\lambda}$ falls outside the capacity region, the EGS cannot support it under any scheduling algorithm, leading to unpredictable performance. The goal of moderating the rate vector through the Rate Control Protocol (RCP) is twofold: first, to keep it within the capacity region, and second, to maximize resource utilization by saturating the capacity region, thus fully leveraging the potential of the EGS to facilitate entanglement generation.

Theorem 1 (Capacity Region)**.** *Let x_s be the maximum number of resources that can be allocated to a session s per time slot. For each resource, p_{gen} is the probability that a communication session allocated the resource for one time slot will successfully create an entangled pair. The capacity region of an EGS with R resources is the set of rate vectors $\boldsymbol{\lambda} \in \text{Int}\mathscr{C}$, where $\mathscr{C}$ is defined as:*

$$\mathscr{C} = \{\boldsymbol{\lambda} : \boldsymbol{\lambda} \geq \mathbf{0},\ \sum_s \lambda_s \leq \lambda_{\text{EGS}},\ \textit{and}\ \lambda_s \leq \lambda_{\text{gen},s}^{\max}\ \forall\ s \in S\}, \tag{2.9}$$

$\lambda_{\text{EGS}} = R \cdot p_{\text{gen}}$ and $\lambda_{\text{gen},s}^{\max} = x_s \cdot p_{\text{gen}}$. Moreover, maximum weight scheduling (Definition 6) is throughput optimal and supports any rate vector $\boldsymbol{\lambda} \in \text{Int}\mathscr{C}$. For proof, see Section 2.6.

The first requirement of $\mathscr{C}$ states that all request rate vectors must be positive, meaning every component of the rate vector must be positive or zero ($\boldsymbol{\lambda} \geq \mathbf{0} \Leftrightarrow \lambda_s \geq 0\ \forall s \in S$). The second requirement enforces that the total rate of entanglement requested from the EGS, $\sum_s \lambda_s$, cannot exceed the total average service rate of the EGS, $R \cdot p_{\text{gen}}$. The final requirement states that the request rate λ_s of any communication session s must not exceed the maximum average service rate that can be allocated to the communication session, $x_s \cdot p_{\text{gen}}$.

2.2.3. CONSTRAINTS

We assume that there are two types of constraints on the sequence of target rates set by a session. The first is a minimum rate of entanglement generation $\lambda_s^{\min}$; below this rate, session s cannot obtain sufficient entangled pairs within a short enough period of time in order to enable its target application. The second constraint $\lambda_u \; \forall u \in U$ is an upper limit on the rate at which each node u can generate and/or make use of entanglement across all of the sessions that it is involved in. This parameter can capture a range of technical limitations of the quantum nodes, including a limited rate of entanglement generation or a limited speed of writing generated entanglement to memory, hence temporarily decreasing the availability of the node for engaging in further entanglement generation immediately following the successful production of a pair.

2.2.4. MATHEMATICAL BACKGROUND

In this section we briefly summarize and motivate some relevant mathematical techniques from optimization theory. See [25] for a thorough coverage of the topic. The method of Lagrange multipliers is a tool for solving constrained optimization problems. Consider the problem of maximizing a continuously differentiable objective function $F(\boldsymbol{z}) : \mathbb{R}^n \mapsto \mathbb{R}$ over the domain $Z \subseteq \mathbb{R}^n$ subject to a set of m inequality constraint functions, $\{g_m(\boldsymbol{z}) \leq 0\}_m$. This is referred to as the *primal problem*,

$$\max_{\boldsymbol{z} \in Z} F(\boldsymbol{z}) \tag{2.10}$$

subject to,

$$g_m(\boldsymbol{z}) \leq 0 \; \forall m. \tag{2.11}$$

The problem is said to be *feasible* if there exists a vector $\boldsymbol{z}$ for which (2.11) is satisfied, meaning the constraints of the problem are satisfied. A local maximizer $\boldsymbol{z}^*$ of the function F over the domain Z satisfies the optimality condition [25],

$$\nabla_{z_n} F(\boldsymbol{z}^*)^T (\boldsymbol{z} - \boldsymbol{z}^*) \leq 0 \; \forall \boldsymbol{z} \in Z. \tag{2.12}$$

In the method of Lagrange Multipliers, one introduces a non-negative vector $\boldsymbol{p} = (p_m \; \forall m) \in \mathbb{R}^{+m}$ and the Lagrangian function,

$$L(\boldsymbol{z}, \boldsymbol{p}) = \; F(\boldsymbol{z}) - \sum_m p_m g_m(\boldsymbol{z}).$$

The following theorem motivates a method of solving the primal problem by searching for vectors $\boldsymbol{z}^*$ that satisfy $\nabla_{z_n} L(\boldsymbol{z}^*, \boldsymbol{p}) = 0$, for some vector of Lagrange multipliers $\boldsymbol{p}$.

Theorem 2 (Karush-Kuhn-Tucker (KKT) conditions [25])**.** *Let $\boldsymbol{z}^*$ be a local maximizer of the function F over the domain Z. Suppose that the functions F and $\{g_m(\boldsymbol{z})\}_m$ are continuously differentiable and that the set of constraint gradients $\{\nabla_{z_n} g_m(\boldsymbol{z}^*) | g_m(\boldsymbol{z}^*) = 0\}_m$ is linearly independent. Then, there exists a unique Lagrange*

multiplier vector $\boldsymbol{p}^*$ *with components* p_m^* *such that*

$$\nabla_{z_n} L(\boldsymbol{z}^*, \boldsymbol{p}^*) = 0, \tag{2.13}$$

$$g_m(\boldsymbol{z}^*) \leq 0 \ \forall m, \tag{2.14}$$

$$p_m^* \geq 0 \ \forall m, \tag{2.15}$$

$$\textit{and } p_m^* \ g_m(\boldsymbol{z}^*) = 0 \ \forall m. \tag{2.16}$$

In some cases, it may be computationally challenging to solve the system of equations (2.13) - (2.16). However, in these cases it may be possible to solve a related problem, known as the *dual problem*,

$$\inf_{\boldsymbol{p} \geq \boldsymbol{0}} D(\boldsymbol{p}), \tag{2.17}$$

where the dual function $D(\boldsymbol{p})$ is defined as,

$$D(\boldsymbol{p}) := \sup_{\boldsymbol{z} \in Z} L(\boldsymbol{z}, \boldsymbol{p}). \tag{2.18}$$

Searching for a solution to the dual problem is particularly useful when there is no *duality gap*, meaning that the value of the optimal solution $D(\boldsymbol{p}^*)$ achieved with a solution $\boldsymbol{p}^*$ to the dual problem is equal to the value of the optimal solution $F(\boldsymbol{z}^*)$ achieved with a solution $\boldsymbol{z}^*$ to the primal problem. The following theorem lays out conditions under which there is no duality gap and the existence of a vector of Lagrange multipliers is guaranteed.

Theorem 3 (Strong Duality Theorem [25])**.** *Suppose that the primal problem is feasible and its optimal value* $F(\boldsymbol{z}^*)$ *is finite. Furthermore, suppose the set* Z *is a convex subset of* $\mathbb{R}^n$*, the function* $F(\boldsymbol{z})$ *is concave over* Z*, and the functions* $\{g_m(\boldsymbol{z})\}_m$ *are convex over* Z*. In addition, suppose there exists a vector* $\overline{\boldsymbol{z}} \in Z$ *such that* $g_m(\overline{\boldsymbol{z}}) < 0 \ \forall m$*. Then, there is no duality gap and there exists at least one vector of Lagrange multipliers.*

The assumption in the Strong Duality Theorem, 3 that the feasible region of the domain Z must contain an interior point is commonly known as the Slater constraint qualification [23].

2.3. RCP ALGORITHM

An algorithm moderating competition for EGS resources enables the possibility of introducing a notion of fairness in how resources are allocated amongst competing communication sessions and ensuring that the resources are fully utilized. We consider a situation where the rate vector produced by any such algorithm is constrained by the maximum service rate of the switch, as described by the capacity region $\mathscr{C}$, as well as the node or user level constraints described by $\lambda_u \ \forall u$ and $\lambda_s^{\min} \ \forall s$. In the framework of NUM, we pose an optimization problem where each communication session s is associated with a utility function $f_s(\lambda_s(t_n)) : \mathbb{R} \mapsto \mathbb{R}$, which encodes the benefit s derives from the rate vector $\boldsymbol{\lambda}(t_n)$. We apply the theory of Lagrange multipliers and Lagrangian duality (see [25] for detailed coverage) to formulate and analyze the optimization problem. We then derive the RCP (Algorithm 1) as the solution to this problem.

The primal problem is to maximize the aggregate utility or the total benefit that users derive from the EGS by maximizing the sum of the utility functions, including the constraints by the use of Lagrange multipliers. The dual problem is to determine an optimal vector of Lagrange multipliers. In the case where there is no duality gap [25], a solution to the dual problem is equivalent to a solution of the primal problem. The vector of Lagrange multipliers $\boldsymbol{p}(t_{n+1}) = \big(p_c(t_n), p_u(t_n)\ \forall u\big) \in \mathbb{R}^{+(1+N)}$, with components for the processor and each node, is denoted as the price vector in our algorithm and serves as a measure of the competition for resources amongst the communication sessions. Define $S(u) := \{s : u \in s\} \subseteq S$ to be the subset of communication sessions in which node u participates. In each communication session one node is designated to communicate demand to the switch and the other node is secondary (see Definition 3). Note that $u \in s \Leftrightarrow s \in S(u)$. The feasible rate region of the communication session s is,

$$\Lambda_s := \{\lambda_s : \lambda_s^{\min} \leq \lambda_s \leq \lambda_{\text{gen},s}^{\max}\}\ \forall\ s, \tag{2.19}$$

and the feasible region for a rate vector $\boldsymbol{\lambda}$ is,

$$\Lambda = \bigcup_s \Lambda_s. \tag{2.20}$$

We make the following two assumptions on the utility function f_s of each communication session s:

A1: On the interval $\Lambda_s = [\lambda_s^{\min}, \lambda_{\text{gen},s}^{\max}]$ the utility functions f_s are increasing, strictly concave, and twice continuously differentiable;

A2: The curvatures of all f_s are bounded away from zero on Λ_s. For some constant $\alpha_s > 0$,

$$-f_s^{''}(\lambda_s) \geq \frac{1}{\alpha_s} > 0\ \forall\ \lambda_s \in \Lambda_s.$$

To ensure feasibility and satisfy the Slater constraint qualification [25], in addition to assumptions A1 and A2 it is necessary that the rate vector with components equal to the minimal rates of each communication session is an interior point of the constraint set,

$$\sum_s \lambda_s^{\min} < \lambda_{\text{EGS}}; \tag{2.21}$$

$$\sum_{s \in S(u)} \lambda_s^{\min} < \lambda_u\ \forall\ u. \tag{2.22}$$

Algorithm 1: Rate Control Protocol (RCP)

Processor's Algorithm: At times $t_n = 1,\ 2, \cdots$, the processor:

1. receives rates $\lambda_s(t_n)$ from all communication sessions $s \in S$;

2. computes a new central price,

$$p_c(t_{n+1}) = \Big[\frac{1}{\lambda_{\text{EGS}}}\sum_s q_s(t_n) + \theta_c\Big(\sum_s \lambda_s(t_n) - \lambda_{\text{EGS}}\Big)\Big]^+, \tag{2.23}$$

where θ_c is a constant step-size for the central price;

3. broadcasts the new central price $p_c(t_{n+1})$ to all communication sessions $s \in S$.

Network Node u's Algorithm: At times $t_n = 1, 2, \cdots$, network node u:

1. marks the subset of communication sessions $\text{COMM}(u) \subseteq S(u)$ involving node u for which it is the designated communication node;

2. receives from every secondary node u' the price $p_{u'}(t_n)$ for each communication session $s = (u, u') \in \text{COMM}(u)$;

3. computes a new node price,

$$p_u(t_{n+1}) = \Big[\frac{1}{\lambda_u}\sum_{s\in S(u)} q_s(t_n) + \theta_u\Big(\sum_{s\in S(u)} \lambda_s(t_n) - \lambda_u\Big)\Big]^+, \ \forall\ u, \tag{2.24}$$

where $\theta_u\ \forall u$ is a constant step-size for each node, which may be fixed or differ from node to node;

4. communicates the new price $p_u(t_{n+1})$ to the communication node from every communication session $s \in S(u) \setminus \text{COMM}(u)$ in which u is a secondary node;

5. receives from the switch the central price $p_c(t_{n+1})$;

6. computes the new rate for every communication session $s \in \text{COMM}(u)$,

$$\lambda_s(t_{n+1}) = \left[\Big(\frac{\mathrm{d}f_s}{\mathrm{d}\lambda_s}\Big)^{-1}\big(\boldsymbol{p}(t_{n+1})\big)\right]_{\lambda_s^{\min}}^{\lambda_{\text{gen},s}^{\max}}, \tag{2.25}$$

where $[z]_m^M = \max\big(\min(z,\ M), m\big)$ and $\boldsymbol{p}(t_i) = \big(p_c(t_i),\ p_u(t_i)\ \forall\ u\big)$ is the vector of prices pertaining to time slot t_i;

7. communicates the new rate $\lambda_s(t_{n+1})$ to the EGS processor, for every communication session $s \in \text{COMM}(u)$.

2.3.1. DERIVATION

Formally, the RCP yields rate vectors which solve the *Primal Problem:*

$$\max_{\boldsymbol{\lambda}\in\Lambda} F(\boldsymbol{\lambda}) := \sum_s f_s(\lambda_s), \tag{2.26}$$

subject to,

$$\sum_s \lambda_s \leq \lambda_{\text{EGS}} \tag{2.27}$$

$$\sum_{s \in S(u)} \lambda_s \leq \lambda_u \; \forall u. \tag{2.28}$$

The Lagrangian, which includes the constraints (2.27), (2.28) with a vector of Lagrange multipliers $\boldsymbol{p} = (p_c, p_u \; \forall u) \geq \mathbf{0}$ together with the objective function (2.26), is given by

$$L(\boldsymbol{\lambda}, \boldsymbol{p}) = \sum_s f_s(\lambda_s) - p_c\Big(\sum_s \lambda_s - \lambda_{\text{EGS}}\Big) - \sum_u p_u\Big(\sum_{s \in S(u)} \lambda_s - \lambda_u\Big). \tag{2.29}$$

We identify that the problem is separable in the communication sessions, S, and re-write the Lagrangian in separable form,

$$L(\boldsymbol{\lambda}, \boldsymbol{p}) = \sum_s l_s(\lambda_s) + p_c \lambda_{\text{EGS}} + \sum_u p_u \lambda_u, \tag{2.30}$$

where $l_s(\lambda_s)$ is defined as

$$l_s(\lambda_s) := f_s(\lambda_s) - \lambda_s p_c - \lambda_s \sum_{u \in s} p_u,$$

and we make use of the equivalence

$$\sum_u p_u \sum_{s \in S(u)} \lambda_s = \sum_s \lambda_s \sum_{u \in s} p_u.$$

A rate vector $\boldsymbol{\lambda}^*$ is a local maximum of (2.26) if it satisfies the optimality condition [25],

$$\boldsymbol{\nabla}_{\lambda_s} F(\boldsymbol{\lambda}^*)^T (\boldsymbol{\lambda} - \boldsymbol{\lambda}^*) \leq 0 \; \forall \boldsymbol{\lambda} \in \; \Lambda. \tag{2.31}$$

If moreover $F(\boldsymbol{\lambda})$ is concave over Λ, then (2.31) is also sufficient for $\boldsymbol{\lambda}^*$ to maximize $F(\boldsymbol{\lambda})$ over Λ [25] (it is also a global maximum).

To obtain a $\boldsymbol{\lambda}^*$ satisfying both the optimality condition (2.31) and the constraints (2.27), (2.28) we set the gradient with respect to rate of each communication session of the Lagrangian to zero,

$$\boldsymbol{\nabla}_{\lambda_s} L = \sum_s \frac{\mathrm{d} l_s(\lambda_s)}{\mathrm{d}\lambda_s} = 0.$$

The maximization in the primal problem (2.26) is constrained to the feasible rate region defined by (2.19), (2.20). To restrict solutions to the problem domain, any $\tilde{\boldsymbol{\lambda}}^* \notin \Lambda$ is projected component-wise so that $\tilde{\lambda}_s^* \mapsto \lambda_S^* \in \Lambda_s \; \forall s$. With the assumptions in (2.21), (2.22) there exists at least one set of Lagrange multipliers [25]. In terms of a given vector of Lagrange multipliers $\boldsymbol{p}$, an optimal rate vector $\boldsymbol{\lambda}^*$ satisfies,

$$\lambda_s^* = \left[\left(\frac{\mathrm{d} f_s}{\mathrm{d}\lambda_s}\right)^{-1}(\mathbf{p})\right]_{\lambda_s^{\min}}^{\lambda_{\text{gen},s}^{\max}} \; \forall \; s, \tag{2.32}$$

where $[z]_m^M = \max\Big(\min\big(z, \; M\big), m\Big)$. To obtain a $\boldsymbol{\lambda}^*$, it remains to obtain a vector of Lagrange multipliers.

An optimal vector $\boldsymbol{p}^*$ of Lagrange multipliers is a solution to the *Dual Problem:* Select $\boldsymbol{p} = (p_c,\ p_u\ \forall\ u)$ so as to achieve,

$$\inf_{\boldsymbol{p}\geq\mathbf{0}} D(\mathbf{p}), \tag{2.33}$$

where the dual objective function $D(\mathbf{p})$ is defined as,

$$D(\mathbf{p}) = \sup_{\boldsymbol{\lambda}\in\Lambda} L(\boldsymbol{\lambda}, \boldsymbol{p}). \tag{2.34}$$

With assumptions A1, A2 and (2.21, 2.22), the problem satisfies the Slater constraint qualification and has no *duality gap* [25], meaning a solution to the dual problem is also a solution to the primal problem. Define $\boldsymbol{\lambda}^*$ to be a rate vector that maximizes $L(\boldsymbol{\lambda},\ \boldsymbol{p})$. A vector of Lagrange multipliers $\boldsymbol{p}^*$ is an optimal solution to the dual problem if it satisfies the optimality condition,

$$\boldsymbol{\nabla}_p D(\boldsymbol{p}^*)^T(\boldsymbol{p}-\boldsymbol{p}^*) \geq 0\ \forall \boldsymbol{p} \geq 0. \tag{2.35}$$

Gradient projection is a type of algorithm where in order to solve an optimization problem such as the dual problem, (2.33), with respect to a vector $\boldsymbol{p}$, one starts by selecting some initial vector $\boldsymbol{p}(0)$ and iteratively adjusting $\boldsymbol{p}(t_n) \mapsto \boldsymbol{p}(t_{n+1})$ by making steps in the opposite direction of the gradient of the objective function. We introduce a vector of step-sizes $\boldsymbol{\theta} = (\theta_c, \theta_u\ \forall u) \in \mathbb{R}^{1+N}$. The components of $\boldsymbol{\nabla}_p D(\boldsymbol{p})$ are,

$$\frac{\partial D(\boldsymbol{p})}{\partial p_c} = -\Big(\sum_s \lambda_s^* - \lambda_{\text{EGS}}\Big); \tag{2.36}$$

$$\frac{\partial D(\boldsymbol{p})}{\partial p_u} = -\Big(\sum_{s\in S(u)} \lambda_s^* - \lambda_u\Big)\ \forall\ u. \tag{2.37}$$

An implementation of the gradient projection algorithm is to iteratively adjusting the Lagrange multipliers according to,

$$p_c(t_{n+1}) = \big[p_c(t_n) + \theta_c\big(\sum_s \lambda_s^*(t_n) - \lambda_{\text{EGS}}\big)\big]^+; \tag{2.38}$$

$$p_u(t_{n+1}) = \big[p_u(t_n) + \theta_u\big(\sum_{s\in S(u)} \lambda_s^*(t_n) - \lambda_u\big)\big]^+,\ \forall\ u, \tag{2.39}$$

where $\lambda_s^*(t_n) = \lambda_s^*\big(\boldsymbol{p}(t_n)\big)$ is given by inputting the vector of Lagrange multipliers in (2.32). An implementation of the algorithm necessitates identifying parameters in the system that correspond to the components of the vector of Lagrange multipliers. We note that the centralized price $p_c(t_n)$ and the user prices $p_u(t_n)\ \forall\ u$, have respectively, the same dynamics as the total queue lengths and the sum total of the session queue lengths in

which user u participates (2.7). Therefore, we make the following identifications,

$$p_c(t_n) \leftrightarrow \frac{1}{\lambda_{\text{EGS}}} \sum_s q_s(t_n);$$
$$p_u(t_n) \leftrightarrow \frac{1}{\lambda_u} \sum_{s \in S(u)} q_s(t_n) \; \forall u.$$

Note that these identifications are not unique, since the only strict criteria on the identification is that the queue dynamics generated by (2.7) match the dynamics of (2.38) and (2.39), whereas the scaling is arbitrary. For more information on the interpretation of Lagrange multipliers as prices in communication networks, see [22, 23].

2.3.2. Convergence

The RCP is a gradient projection algorithm with constant step-sizes from the vector $\boldsymbol{\theta} \in \mathbb{R}^{1+N} = (\theta_c, \theta_u \; \forall u)$. Establishing that the algorithm converges is crucial to ensure that it yields solutions that effectively address the problem it is designed to solve. To establish convergence, we follow a similar treatment as in [26].

Theorem 4 (RCP Convergence)**.** *Suppose assumptions A1 and A2 and the constraints (2.21, 2.22) are satisfied and each of the the step-sizes $\theta_r \in \{\theta_c, \theta_u \; \forall u\}$ satisfies $\theta_r \in (0, 2/\overline{\alpha}|S|)$, where $\overline{\alpha} = \max_{s \in S} \alpha_s$ with α_s the curvature bound of assumption A2, and $|S|$ is the number of communication sessions. Then, starting from any initial rate $\boldsymbol{\lambda}(0) \in \Lambda$ and price $\boldsymbol{p}(0) \geq \mathbf{0}$ vectors, every accumulation point $\left(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{p}}\right)$ of the sequence over time slots $\{\left(\boldsymbol{\lambda}(t_n), \boldsymbol{p}(t_n)\right)\}_{t_n}$ generated by the RCP is primal-dual optimal. For proof, see Section 2.6.*

2.4. Case Study

To illustrate use of the RCP we associate a log utility function with each session,

$$f_s(\lambda_s) = \log(\lambda_s) \; \forall \; s \in S. \tag{2.40}$$

Log utility functions are suitable when throughput is the target performance metric, and a set of sessions all employing log utility functions will have the property of proportional fairness. In such a system, if the proportion by which one session rate changes is positive, there is at least one other session for which the proportional change is negative [23]. For compatibility with Theorem 4 note that log utility functions satisfy A1, and A2 is satisfied with $\alpha_s = (\lambda_{\text{gen},s}^{\max})^2 \; \forall s$.

Although the convergence theorem only guarantees asymptotic convergence of the sequence $\{\left(\boldsymbol{\lambda}(t_n), \boldsymbol{p}(t_n)\right)\}_{t_n}$ to an optimal rate-price pair $\left(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{p}}\right)$, in any realization of an EGS one expects that the convergence time $\Delta\tau$, the number of time slots that the RCP must run before convergence is attained, is finite. In addition, it is practically relevant to characterize the tightness of convergence δ, or the maximum size of fluctuations about the optima.

If an EGS is connected to N nodes, there are $|S|_{\max} = \binom{N}{2}$ possible sessions. We assume that in a real network not all pairs of users require shared entanglement. In Fig. 2.2 we numerically investigate the convergence time and tightness of convergence, $(\Delta\tau, \delta)$, for an EGS with $R = 3$ resources and $p_{\text{gen}} = 0.05$ connected to $N = 20$, 50 and 100 users,

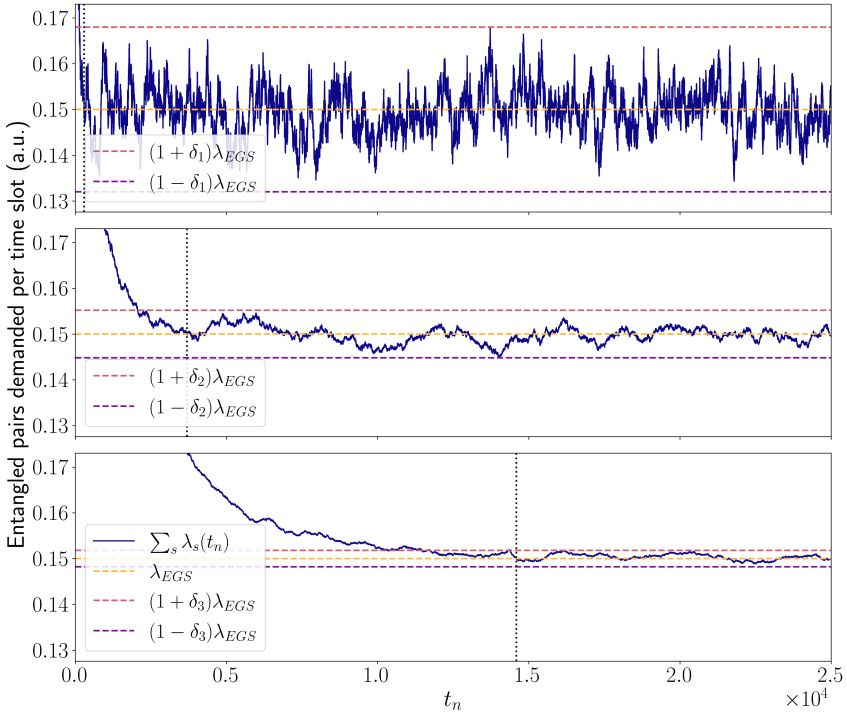


Figure 2.2: The RCP drives the sum of the demanded rates of entanglement generation across all communication sessions, $\sum_s \lambda_s(t_n)$, to converge with respect to the sequence of time slots to the maximum average entanglement generation rate of the EGS, λ_{EGS}. The EGS has $R = 3$ resources, the probability of entanglement generation is $p_{\text{gen}} = 0.05$, and the EGS is connected to $N = 20$ (top), $N = 50$ (middle) and $N = 100$ (bottom) nodes. The total number of communication sessions served are $|S| = 19,\ 123,$ and 495 in the top, middle, and bottom plots, respectively. Black dotted lines indicate the convergence times, $\Delta\tau$. The observed values for the tightness of convergence, δ, are $\delta_1 = 0.12$, $\delta_2 = 0.035$ and $\delta_3 = 0.012$. Step-sizes $(\theta_c, \theta_u\ \forall u)$ were all $1/(40 \cdot \lambda_{\text{EGS}})$.

where the number of sessions is restricted to $|S| = 0.1 \cdot |S|_{\max}$ by randomly sampling 10% of the possible sessions. In these simulations we set $x_s = 1\ \forall s$, and average over 1000 independent runs of the simulation, each using the same set of sessions.

The reported convergence times $\Delta\tau$ are the number of time slots that occur before the sum of demand rates $\sum_s \lambda_s(t_n)$ first crosses the optimal value λ_{EGS}. Reporting of the tightness of convergence, δ, is based on the maximum size of fluctuations of $\sum_s \lambda_s(t_n)$ about λ_{EGS} following $\Delta\tau$. As the number of sessions hosted by an EGS increases, we observe a trade-off between $\Delta\tau$ and δ. When the number of sessions is lower, $\Delta\tau$ is shorter but δ is larger. We have performed additional simulations which indicate that increasing the step size used in the RCP can be used to trade larger δ for somewhat shorter $\Delta\tau$.

If constraint changes occur slowly compared to $\Delta\tau$, Theorem 4 implies that the RCP should re-establish convergence to a new optimal rate and price vector pair, $(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{p}}) \mapsto (\hat{\boldsymbol{\lambda}}',\ \hat{\boldsymbol{p}}')$. In a real EGS system it is possible that the number of available resources will not be static in time, as resources may require periodic downtime for calibration.

The effect of a change in the number of resources $R \mapsto R'$ changes the maximum service rate $\lambda_{\text{EGS}} = R \cdot p_{\text{gen}} \mapsto \lambda'_{\text{EGS}} = R' \cdot p_{\text{gen}}$. To validate the robustness of the algorithm against such constraint changes we simulate EGS systems originally equipped with $R = 3$ resource nodes, where after every 10,000 time-slots one of the resources may either be taken offline for calibration or an offline resource may be returned to service. Fig. 2.3 demonstrates that the RCP successfully re-establishes convergence of $\sum_s \lambda_s(t_n)$ about λ'_{EGS} following these constraint changes in an EGS system connected to $N = 50$ nodes, serving $|S| = 123$ communication sessions.

In Fig. 2.3 we record the sequence of convergence times, $\{\Delta\tau\}$, after each constraint change as the first time-steps where $\sum_s \lambda_s(t_n)$ crosses λ'_{EGS}. To calculate the tightness of convergence, δ, we first calculate the sequence of $\{\delta'\}$, the size of the maximum fluctuations about λ'_{EGS} following each $\Delta\tau'$ and set $\delta = \max(\{\delta'\})$. Notably, every subsequent $\Delta\tau' < \Delta\tau$ and the achieved δ is equal to that observed when there are no changes to the constraint set in Fig. 2.2 (middle plot, δ_2) for an EGS with the same number of nodes, serving the same number of communication sessions. Additional simulations of EGS systems connected to various numbers of nodes ranging from 10 to 100, with random changes to the number of resources after every 10,000 time-steps, suggest that the data in Fig. 2.3 is representative. Specifically, in each case investigated the absolute relative difference,

$$\frac{|\delta - \tilde{\delta}|}{\tilde{\delta}} < 1$$

between the achieved tightness of convergence when there are (δ) and are not ($\tilde{\delta}$) changes to the constraints is less than 1.

The constraints $\{\lambda_u\}_u$ on the capabilities of nodes appear in (2.24) and therefore affect both the prices calculated by the nodes and the rates set by communication sessions in (2.25). Since these constraints limit the total rate at which a node can submit demands summed across all of the communication sessions in which it participates, it is expected that uniform settings of $\{\lambda_u\}_u$ yield rate vectors under the RCP where $\{\lambda_s(t_n)\}_s$ are approximately uniform. In contrast, if the node constraints are non-uniform amongst the nodes, it is expected that the RCP yields rate vectors with larger differences between the rates set by each communication session. In Fig. 2.4 we investigate the effect of different settings for these constraints by plotting the difference between the average maximum $\max_s\{\lambda_s(t_n)\}_s$ and minimum $\min_s\{\lambda_s(t_n)\}_s$ communication session rates yielded by the RCP for two different settings of the constraints. In the first setting, node constraints are set uniformly as $\lambda_u = \big((|S|-1)/2\big) \cdot p_{\text{gen}}\ \forall u$ so that in practice the algorithm functions as if the network node constraints have been removed. In the other setting there are three possible constraint values: a quarter of the nodes sampled at random have $\lambda_u = 1.5 \cdot p_{\text{gen}}$, half of the nodes have $\lambda_u = p_{\text{gen}}$, and a quarter of the nodes have $\lambda_u = 0.5 \cdot p_{\text{gen}}$. Fig. 2.4 confirms that the difference between the average maximum rate and the average minimum rate requested by any session at time-step t_n is one or more orders of magnitude larger when nodes are associated with the non-uniform constraint set. The uniform node constraint setting led to communication sessions updating their rates of demand submission to be nearly uniform across all communication sessions.

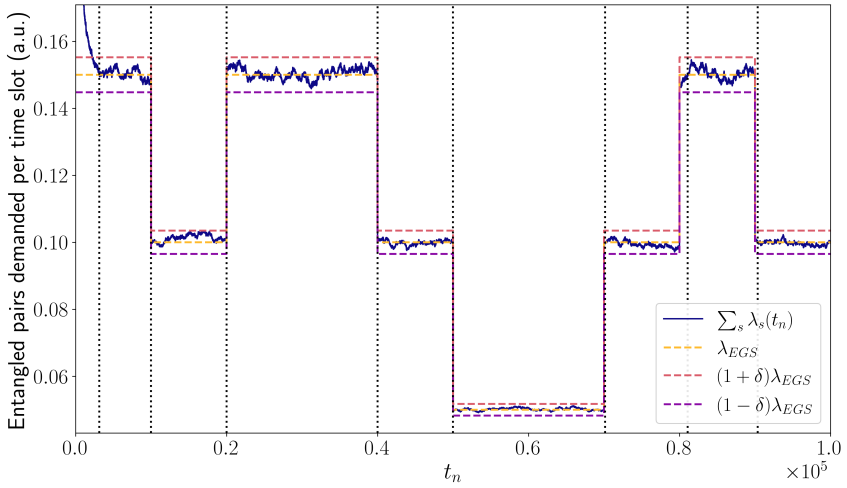


Figure 2.3: In response to changes in the number of resources available at the EGS ($R \rightarrow R'$), the RCP drives the sum of the demanded rates of entanglement generation across all communication sessions, $\sum_s \lambda_s(t_n)$, to converge with respect to the sequence of time slots to the updated maximum average entanglement generation rate of the EGS, $\lambda_{\text{EGS}} = R' \cdot p_{\text{gen}}$. In simulation, an EGS connected to $N = 50$ nodes, serving $|S| = 123$ communication sessions, is originally equipped with $R = 3$ resources. After every 10,000 time-slots, one of the resources may either be taken offline for calibration or an offline node may be returned to service. Black dashed lines indicate the convergence, $\Delta\tau$ calculated for every R' (initially R). We observe and overall tightness of convergence of $\delta = 0.035$, identical to that observed in Fig. 2.3 for the EGS operated with fixed $R = 3$ and with the same N, $|S|$. Step-sizes ($\theta_c, \theta_u \; \forall u$) were all $1/(10 \cdot \lambda_{\text{EGS}})$.

2.5. DISCUSSION

We have presented the first control architecture for an EGS. The architecture is tailored to a simple system model. As a natural extension of this work, a refined version of the control architecture can be developed to suit a more versatile physical model. In the following discussion, we explore considerations for the development of a second generation control architecture.

In this work we assume a demand model in which user generated demands are fully parameterized by a desired rate of entanglement generation. Specifically, every communication session s sets $\lambda_s(t_n)$, updated once per time-slot and specifies the constraint parameter $\lambda_s^{\min}$ which defines the minimum rate of entanglement generation the communication session must receive in order to enable some target application. While this model is mathematically simple, it may not fully address real application requirements on a physical quantum network. Real applications may require the simultaneous existence of a number of entangled pairs, each with some minimum fidelity and it is possible that applications need such packets of pairs to be supplied periodically over a longer application run-time. In the future, it may therefore be relevant to consider a demand model wherein communication sessions submit demands for packets of entanglement generation. A packet would be fully specified by the desired number of entangled pairs, a minimum fidelity for the pairs, some maximum window of time between the generation time of the first and last entangled pair of the request, and possibly some rate at which

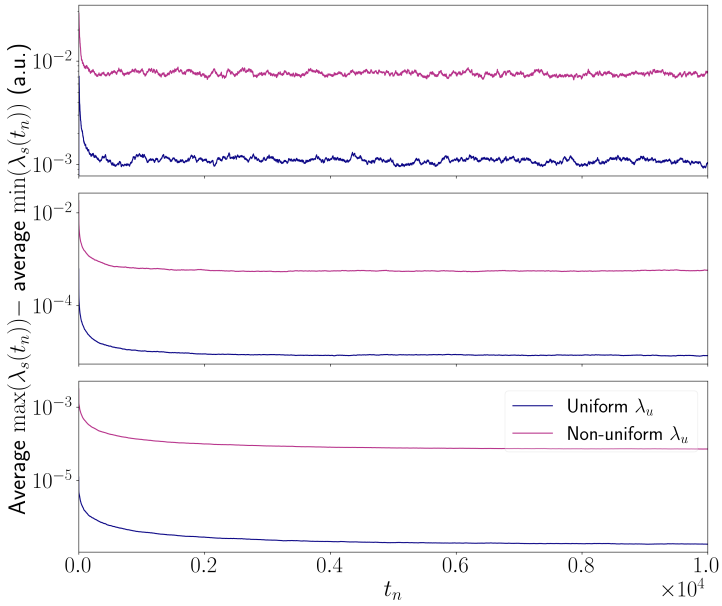


Figure 2.4: Differences between the average maximum rate and average minimum rate requested by any communication session in time-slot t_n, for an EGS connected to $N = 20$ (top), $N = 50$ (middle) and $N = 100$ (bottom) nodes serving $|S| = 19$, 123, and 495 communication sessions, respectively. As described in the main text, nodes are either associated with a uniform and effectively un-restricted set of capabilities or a non-uniform and more restricted set of capabilities. Step-sizes $(\theta_c, \theta_u \; \forall u)$ were all $1/(40 \cdot \lambda_{\mathrm{EGS}})$.

the demand with the preceding parameters should be repeatedly fulfilled.

The discussed model assumes that user controlled nodes can engage in multiple entanglement generation tasks in parallel. We do not impose restrictions on simultaneously scheduling communication sessions. Hence, it is possible for communication sessions s and s' with node $u \in s, s'$ to be scheduled simultaneously. Additionally, we consider the option of assigning multiple resource nodes to a single communication session in any time-slot. Therefore, we consider nodes with an unrestricted number of qubits and independent physical connections to the EGS. A subtlety we do not address here is that allocating multiple resources to a single communication session may require temporal multiplexing in the scheduling of individual entanglement generation attempts, especially when the multiple qubits of a single node are coupled to the physical connection via a single output. Furthermore, for nodes consisting of a single quantum processor, it may not be possible to calibrate the node to simultaneously engage in entanglement generation attempts with multiple partner nodes, even if the node has unlimited qubits. To capture this physical feature, it will be interesting to include the restriction of scheduling only non-overlapping communication sessions in the design of scheduling routines for future EGS control architectures.

The control architecture for an EGS relies on precise timing synchronization. Our model assumes that at both the control and physical layers, all communication sessions can adhere to the time slots defined by the EGS processor. Tight synchronization of timing

is possible at the physical layer, which controls the quantum devices and coordinates the exact timing of entanglement generation attempts. However, tight timing synchronization of any type of classical communication may be a considerable challenge in any real world application. In particular, such coordination is a serious challenge if there are non-uniform communication times between any of the nodes and the EGS or between any of the node pairs. To reduce the timing requirements and possibly make the control architecture delineated here executable on a real-world system, it is possible to consider the processor of the EGS simulating the actions of the nodes. To do so, the processor would locally run the RCP and simulate the generation of demands originating from the user operated nodes by simply adding demands to the queues based on the rates output by the RCP. Such an approach trades the difficulty of timing synchronization for the requirement of increased power of the classical processor at the EGS. To reduce the need for timing synchronization, a second generation architecture may be designed which does not rely on fixed, centrally defined time slots.

2.6. PROOFS

OUTLINE OF GOALS TO PROVE

In this section we will prove two theorems to establish the results quoted in Section 2.3.2. The results are as follows:

1. The capacity region of the EGS is the set of demand arrival rate vectors fully contained in the set $\mathscr{C}$ (2.9) and maximum weight scheduling (Definition 6) supports any rate vector from within $\mathscr{C}$ (Theorem 1). To establish the capacity region, we first prove a proposition stating that any rate vector $\boldsymbol{\lambda} \notin \mathscr{C}$ necessarily results in divergent queues. We then prove a second proposition establishing at once that any rate vector $\boldsymbol{\lambda} \in \text{Int}\mathscr{C}$ is supportable under some scheduling algorithm and that maximum weight scheduling is such a scheduling algorithm. Therefore, we also demonstrate that maximum weight scheduling is throughput optimal.

2. The RCP, Algorithm 1, results in the calculation of a sequence of rate and price vector pairs $\left(\boldsymbol{\lambda}(t_n), \boldsymbol{p}(t_n)\right)$ which converge to optimal solutions $\left(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{p}}\right)$ of the primal and dual problems, defined in Section 2.3 (Theorem 4).

PROOF OF THEOREM 1

First it is to be shown that no rate vector $\boldsymbol{\lambda} \notin \mathscr{C}$ of an EGS with R resources is supportable under any scheduling algorithm.

Proposition 1. *If $\boldsymbol{\lambda} \notin \mathscr{C}$, no scheduling algorithm can support $\boldsymbol{\lambda}$.*

Proof. There are three cases where $\boldsymbol{\lambda} \notin \mathscr{C}$,

1. $\sum_s \lambda_s > R \cdot p_{\text{gen}}$, or

2. $\lambda_{s^*} > x_{s^*} \cdot p_{\text{gen}}$ for some $s^* \in S$.

3. $\boldsymbol{\lambda}$ is not non-negative ($\exists\, \lambda_{s^*} < 0$ for at least some $s^* \in S$).

In the third case, the node pair corresponding to session s^* has set a non-physical rate and the rate must be changed. The proof for case (2) is very similar to case (1) and equations from the first case are re-used or modified to complete the proof of case (2). The main strategy of the proof relies on Definition 7; a rate vector $\boldsymbol{\lambda} \notin \mathscr{C}$ is not supportable if $\boldsymbol{\lambda}$ causes the queue lengths at the EGS processor to diverge with probability 1, regardless of scheduling algorithm. To prove the proposition in each case, it serves to calculate the total queue length.

Proposition 1 (1): Suppose $\sum_s \lambda_s > R \cdot p_{\text{gen}}$. Then, $\exists\, \epsilon > 0$ such that

$$\sum_s \lambda_s \geq R \cdot p_{\text{gen}} + \epsilon. \tag{2.41}$$

Assume that the initial length of each queue is finite. The sum of queue lengths at time step t_{n+1}, $\sum_s q_s(t_{n+1})$ is,

$$\begin{aligned} \sum_s q_s(t_{n+1}) &= \sum_s \left[q_s(t_n) + a_s(t_n) - g_s(t_n) \right]^+ \\ &\geq \sum_s \left(q_s(t_n) + a_s(t_n) - g_s(t_n) \right) \\ &\geq \sum_s \left(q_s(t_1) + \sum_{t_i=t_1}^{t_n} \left(a_s(t_i) - g_s(t_i) \right) \right) \end{aligned} \tag{2.42}$$

where $a_s(t_i)$ is the integer number of demands submitted by communication session s at time step t_i and $g_s(t_i)$ is the integer number of successfully generated entangled pairs between the nodes corresponding to communication session s in time step t_i. The final inequality in (2.42) follows from the previous inequality by repeated application of (2.7). By the strong law of large numbers,

$$\lim_{n\to\infty} \frac{1}{t_n} \sum_{t_i=t_1}^{t_n} a_s(t_i) = \lambda_s \;\forall\; s \in S, \text{ with probability 1.} \tag{2.43}$$

Recall that the number of successfully generated entangled pairs between the nodes corresponding to communication session s at time t_i is a sample from a binomial random process where the number of trials is set by $M_s(t_i)$ and the trial success probability is p_{gen},

$$g_s(t_i) \sim \text{Bin}\big(M_s(t_i),\, p_{\text{gen}}\big).$$

By the strong law of large numbers,

$$\lim_{n\to\infty} \frac{1}{t_n} \sum_{t_i=t_1}^{t_n} g_s(t_i) = M_s(t_n) \cdot p_{\text{gen}}, \text{ with probability 1.} \tag{2.44}$$

Since each feasible schedule satisfies $\sum_s M_s(t_i) \leq R$, it follows from (2.44) that

$$\begin{aligned} \lim_{n\to\infty} \frac{1}{t_n} \sum_{t_i=t_1}^{t_n} \sum_s g_s(t_i) &= \sum_s \lim_{n\to\infty} \frac{1}{t_n} \sum_{t_i=t_1}^{t_n} g_s(t_i) \\ &\leq R \cdot p_{\text{gen}}, \end{aligned} \tag{2.45}$$

where we use the distribution property of limits, which is possible because the individual limits (2.44) exist. Finally, by assumption (2.41) and (2.42, 2.43) and (2.45),

$$\begin{aligned}\lim_{n\to\infty}\frac{1}{t_n}\sum_s q_s(t_{n+1}) &\geq \lim_{n\to\infty}\frac{1}{t_n}\sum_s q_s(t_1)+\lim_{n\to\infty}\frac{1}{t_n}\sum_{t_i=t_1}^{t_n}\Big(\sum_s a_s(t_i)-\sum_s g_s(t_i)\Big)\\ &\geq \sum_s \lambda_s - R\cdot p_{\text{gen}}\\ &\geq R\cdot p_{\text{gen}}+\epsilon - R\cdot p_{\text{gen}}\\ &\geq \epsilon. \end{aligned} \tag{2.46}$$

Therefore, with probability 1, $\sum_s q_s(t_n)\to\infty$ as $n\to\infty$, so $\boldsymbol{\lambda}$ is not supportable, regardless of scheduling algorithm.

Proposition 1 (2): Suppose that $\lambda_{s^*} > x_{s^*}\cdot p_{\text{gen}}$ for some $s^*\in S$. Then, $\exists\,\epsilon>0$ such that,

$$\lambda_{s^*}\geq x_{s^*}\cdot p_{\text{gen}}+\epsilon. \tag{2.47}$$

In this case, we show that $\boldsymbol{\lambda}$ is not supportable by proving that the queue $q_{s^*}(t_i)$ of demands associated with communication session s^* diverges for large t_i. Recall (2.44) and note $M_s(t_i)\leq x_s\ \forall\ s,\ \forall\ t_i$. This inequality describes that a maximum of x_s heralding stations can be allocated any communication session s in t_i. With this restriction, (2.44) becomes,

$$\lim_{n\to\infty}\frac{1}{t_n}\sum_{t_i=t_1}^{t_n} g_s(t_i)\leq x_s\cdot p_{\text{gen}}\ \forall s. \tag{2.48}$$

Combining assumption (2.47) using (2.43), (2.48), and making repeated use of (2.7),

$$\begin{aligned}&\lim_{n\to\infty}\frac{1}{t_n}q_{s^*}(t_{n+1})\\ &\quad\geq \lim_{n\to\infty}\frac{1}{t_n}q_{s^*}(t_1)\\ &\qquad+\lim_{n\to\infty}\frac{1}{t_n}\sum_{t_i=t_1}^{t_n}\big(a_{s^*}(t_i)-g_{s^*}(t_i)\big)\\ &\quad\geq \lambda_{s^*}-x_{s^*}\cdot p_{\text{gen}}\\ &\quad\geq x_{s^*}\cdot p_{\text{gen}}+\epsilon-x_{s^*}\cdot p_{\text{gen}}\\ &\quad\geq \epsilon. \end{aligned} \tag{2.49}$$

Therefore, with probability 1, $q_{s^*}(t_{n+1})\to\infty$ as $n\to\infty$. Hence $\boldsymbol{\lambda}$ is not supportable. □

Proposition 1 proved that rate vectors $\boldsymbol{\lambda}\notin\mathscr{C}$ are not in the capacity region of the EGS. To finish proving $\mathscr{C}$ is the capacity region of the EGS (Theorem 1), it is necessary to prove that any rate vector $\boldsymbol{\lambda}\in\mathscr{C}$ is supportable under some scheduling algorithm. To do so, we prove that the specific scheduling algorithm of Maximum Weight Scheduling (Definition 6) supports all arrival rate vectors fully contained in $\mathscr{C}$.

Proposition 2. *Maximum Weight scheduling can support any arrival rate vector* $\boldsymbol{\lambda}$ *for which* $\exists \epsilon > 0$ *such that* $(1+\epsilon)\boldsymbol{\lambda} \in \mathscr{C}$.

Modelling a queue vector as a Markov chain is a standard tool in queuing theory [23]. This approach makes it possible to take advantage of the many strong analytic results on the behaviour of Markov chains, which can then be used to make statements about the queue vector. The vector $\boldsymbol{q}(t_n) = \left(q_s(t_n)\ \forall\ s\right)$ of queued demands from each communication session maintained in the processor at t_n can be modeled as a Markov chain, with transitions given by (2.7). An irreducible Markov chain has the property that any state i of the chain is reachable from any other state j. A positive recurrent Markov chain has the property that from any state i, the expectation value of the time it will take to re-visit any other state j is finite. A queue vector, with specified dynamics, that can be modelled as an irreducible Markov chain with the property of positive recurrence will not diverge (i.e. is guaranteed to remain a finite queue) [23]. The dynamics of such a queue vector are fixed by the arrival rate vector and the scheduling routine, therefore if a queue vector can be modelled as a positive recurrent Markov chain, the arrival rate vector is supportable by the scheduling routine. To prove Proposition 2 we demonstrate that the queue vector is an irreducible Markov chain and use the Foster-Lyapunov Theorem to prove that whenever $\boldsymbol{\lambda}$ lies strictly within $\mathscr{C}$ the Markov chain is also positive recurrent. An equivalent statement is that all rate vectors lying strictly within $\mathscr{C}$ are supportable by some scheduling algorithm.

Theorem 5 (Foster-Lyapunov Theorem [23])**.** *Let* $\{X_k\}$ *be an irreducible Markov chain with a state space* $\mathscr{S}$*. Suppose that there exists a function* $V : \mathscr{S} \to \mathbb{R}^+$ *and a finite set* $\mathscr{B} \subseteq \mathscr{S}$ *satisfying the following conditions:*

1. $\mathbb{E}[V(X_{k+1}) - V(X_k) | X_k = x] \leq -\epsilon$ *if* $x \in \mathscr{B}^c$*, for some* $\epsilon > 0$*, and*
2. $\mathbb{E}[V(X_{k+1}) - V(X_k) | X_k = x] \leq A$ *if* $x \in \mathscr{B}$*, for some* $A < \infty$.

Then the Markov chain $\{X_k\}$ *is positive recurrent.*

Proof of Proposition 2. First we establish that the queue vector, $\boldsymbol{q}(t_i)\ \forall\ t_i$ is an irreducible Markov chain. The queue vector, $\boldsymbol{q}(t_i)$ is a Markov chain with state space

$$\mathscr{S} = \{\boldsymbol{q} : \boldsymbol{q} \text{ is reachable from } \mathbf{0} \text{ under the given scheduling algorithm}\}.$$

Assume that $\boldsymbol{q}(t_1)$ is finite and $\boldsymbol{q}(t_1) \in \mathscr{S}$. It follows from the definition of $\mathscr{S}$ that $\boldsymbol{q}(t_i) \in \mathscr{S}\ \forall t_i$ if $\boldsymbol{q}(t_1) \in \mathscr{S}$. Irreducibility of $\boldsymbol{q}(t_i)\ \forall t_i$ requires that any state $\boldsymbol{q}(t_j)$ is reachable from any other state $\boldsymbol{q}(t_i)$. By the definition of the state space $\mathscr{S}$, it suffices to demonstrate that from $\boldsymbol{q}(t_i)$, the Markov chain can always return to $\mathbf{0}$. Under Maximum Weight scheduling (Definition 6), the processor always serves $k(t_i)$ demands per time-slot, where

$$k(t_i) = \max\{k\ :\ k \leq R \text{ and } k \leq \sum_s \min\left(|q_s(t_i)|,\ x_s\right)\},$$

where $|q_s(t_i)|$ is the number of demands in the queue for session s in time-slot t_i and x_s is the maximum number of resource modules that can be allocated communication session s per time-slot. Hence when $\boldsymbol{q}(t_i)$ is non-zero, at least one demand and up to R demands

are served per time-slot. Therefore, from any $\boldsymbol{q}(t_i) \in \mathscr{S}$, $\boldsymbol{q}(t_{i+l}) = \mathbf{0}$ is reachable from $\boldsymbol{q}(t_i)$ in $l \in \{\lceil \frac{|\boldsymbol{q}(t_i)}{R} \rceil, \lceil \frac{|\boldsymbol{q}(t_i)}{R} \rceil + 1, \cdots, |\boldsymbol{q}(t_i)|\}$ time steps, where $|\boldsymbol{q}(t_i)| := \sum_s |q_s(t_i)|$. Since any other $\boldsymbol{q}(t_j) \in \mathscr{S}$ is then reachable from $\mathbf{0}$, it follows that $\boldsymbol{q}(t_i)$ is irreducible. To prove that $\boldsymbol{\lambda}$ is supportable, it suffices to demonstrate that $\boldsymbol{q}(t_i)$ is positive recurrent.

Define the Lyapunov function

$$L\big(\boldsymbol{q}(t_i)\big) = \frac{1}{2}\sum_s q_s^2(t_i). \tag{2.50}$$

To apply the the Foster-Lyapunov theorem (5), the key quantity is the drift of $L\big(\boldsymbol{q}(t_i)\big)$. Using the queue update dynamics (2.7), the drift can be expanded as

$$\begin{aligned}
&L\big(\boldsymbol{q}(t_{i+1})\big) - L\big(\boldsymbol{q}(t_i)\big) \\
&= \frac{1}{2}\sum_s \Big(\big[q_s(t_i) + a_s(t_i) - g_s(t_i)\big]^+\Big)^2 - \frac{1}{2}\sum_s q_s^2(t_i) \\
&\le \frac{1}{2}\sum_s \big(q_s(t_i) + a_s(t_i) - g_s(t_i)\big)^2 - \frac{1}{2}\sum_s q_s^2(t_i) \\
&= \frac{1}{2}\sum_s \big(a_s(t_i) - g_s(t)\big)^2 \\
&\quad + \sum_s q_s(t_i)\big(a_s(t_i) - g_s(t_i)\big).
\end{aligned} \tag{2.51}$$

Taking the conditional expectation of the Lyapunov drift with respect to the randomness of arrivals and the probabilistic success of scheduled demands,

$$\begin{aligned}
&\mathbb{E}\big[L\big(\boldsymbol{q}(t_{i+1})\big) - L\big(\boldsymbol{q}(t_i)\big) \mid \boldsymbol{q}(t_i) = \tilde{\boldsymbol{q}}\big] \\
&\quad \le \frac{1}{2}\sum_s \mathbb{E}\big[\big(a_s(t_i) - g_s(t_i)\big)^2 \mid \boldsymbol{q}(t_i) = \tilde{\boldsymbol{q}}\big] \\
&\quad + \sum_s \mathbb{E}\big[q_s(t_i)\big(a_s(t_i) - g_s(t_i)\big) \mid \boldsymbol{q}(t_i) = \tilde{\boldsymbol{q}}\big],
\end{aligned} \tag{2.52}$$

where $\tilde{\boldsymbol{q}} \in \mathscr{S}$ is a particular queue vector.

Using $\big(a_s - g_s\big)^2 \le a_s^2 + {g_s}^2$ and the linearity of expectation, the first term of the conditional expectation can be re-written,

$$\begin{aligned}
&\mathbb{E}\big[\sum_s \big(a_s(t_i) - g_s(t_i)\big)^2 \mid \boldsymbol{q}(t_i) = \tilde{\boldsymbol{q}}\big] \le \\
&\quad \sum_s \mathbb{E}\big[a_s^2(t_i) \mid \boldsymbol{q}(t_i) = \tilde{\boldsymbol{q}}\big] + \sum_s \mathbb{E}\big[{g_s}^2(t_i) \mid \boldsymbol{q}(t_i) = \tilde{\boldsymbol{q}}\big]
\end{aligned} \tag{2.53}$$

Recall that $g_s(t_i) \le M_s(t_i) \le x_s \ \forall\ s, \ \forall\ t_i$. Hence,

$$\sum_s \mathbb{E}\big[{g_s}^2(t) \mid \boldsymbol{q}(t) = \tilde{\boldsymbol{q}}\big] \le \sum_s x_s^2. \tag{2.54}$$

Define the variance in the arrivals to the queue of session s, $\sigma_s^2 := \mathrm{Var}[a_s(t_i)]$. Then, noting that the arrivals are independent of the state of the queues, using the definition of variance and $\mathbb{E}[a_s(t_i)] = \lambda_s$,

$$\mathbb{E}[a_s^2(t_i) \mid \boldsymbol{q}(t_i) = \tilde{\boldsymbol{q}}] = \mathbb{E}[a_s^2(t_i)] = \sigma_s^2 + \lambda_s^2 \tag{2.55}$$

Together (2.54) and (2.55) bound the first term of (2.52),

$$\frac{1}{2}\sum_s \mathbb{E}\big[\big(a_s(t_i) - g_s(t_i)\big)^2 \mid \boldsymbol{q}(t_i) = \tilde{\boldsymbol{q}}\big] \leq \frac{1}{2}\sum_s \big(\sigma_s^2 + \lambda_s^2 + x_s^2\big) =: B$$

Then (2.52) is,

$$\begin{aligned}\mathbb{E}\big[L\big(\boldsymbol{q}(t_{i+1})\big) - L\big(\boldsymbol{q}(t_i)\big) \mid \boldsymbol{q}(t_i) = \tilde{\boldsymbol{q}}\big] &\leq B + \sum_s \mathbb{E}\big[q_s(t_i)\big(a_s(t_i) - g_s(t_i)\big) \mid \boldsymbol{q}(t_i) = \tilde{\boldsymbol{q}}\big] \\ &= B + \sum_s \tilde{q}_s\big(\lambda_s - \mathbb{E}\big[g_s(t_i) \mid \boldsymbol{q}(t_i) = \tilde{\boldsymbol{q}}\big]\big).\end{aligned} \tag{2.56}$$

Recall that the conditional expectation of the Lyapunov drift is taken with respect to the randomness of the arrival processes as well as the success of scheduled demands. The schedule selected for a given time-slot depends on the queues, but the success of any scheduled demand does not. The conditional expectation of pair production for communication session s can be re-written as,

$$\mathbb{E}[g_s(t_i) \mid \boldsymbol{q}(t_i) = \tilde{\boldsymbol{q}}] = p_{\text{gen}} \cdot \mathbb{E}[M_s(t_i) \mid \boldsymbol{q}(t_i) = \tilde{\boldsymbol{q}}]. \tag{2.57}$$

Recall that $\boldsymbol{M}$ denotes the schedule decided under the maximum weight scheduling policy, 6. Allow $\tilde{\boldsymbol{M}}$ to denote a schedule that is decided by any other scheduling policy. It follows from Definition 6 that,

$$\sum_s \tilde{q}_s \cdot \mathbb{E}[M_s(t_i) \mid \boldsymbol{q}(t_i) = \tilde{\boldsymbol{q}}] \geq \sum_s \tilde{q}_s \cdot \mathbb{E}[\tilde{M}_s(t_i) \mid \boldsymbol{q}(t_i) = \tilde{\boldsymbol{q}}]. \tag{2.58}$$

Consider a scheduling policy $\tilde{\boldsymbol{M}}$ which schedules each session at a rate of $\frac{\lambda_s + \epsilon}{p_{\text{gen}}}$ (this is possible since, by assumption, $(1+\epsilon)\boldsymbol{\lambda} \in \mathscr{C}$). Such a scheduling policy is aware of the demand arrival rates to each queue but is not demand based (i.e. it does not use queue information in deciding the schedule). Hence,

$$\begin{aligned}\sum_s \tilde{q}_s \cdot \mathbb{E}[\tilde{M}_s(t_i) \mid \boldsymbol{q}(t_i) = \tilde{\boldsymbol{q}}] &= \sum_s \tilde{q}_s \cdot \mathbb{E}[\tilde{M}_s(t_i)] \\ &= \sum_s \tilde{q}_s\Big(\frac{\lambda_s + \epsilon}{p_{\text{gen}}}\Big).\end{aligned} \tag{2.59}$$

Combining (2.57), (2.58) and (2.59), the conditional expectation of the Lyapunov drift is bounded by,

$$\begin{aligned}\mathbb{E}\big[L\big(\boldsymbol{q}(t_{i+1})\big) - L\big(\boldsymbol{q}(t_i)\big) \mid \boldsymbol{q}(t_i) = \tilde{\boldsymbol{q}}\big] &\leq B + \sum_s \tilde{q}_s\lambda_s - \sum_s \tilde{q}_s \cdot p_{\text{gen}} \cdot \mathbb{E}\big[M_s(t_i) \mid \boldsymbol{q}(t_i) = \tilde{\boldsymbol{q}}\big]\big) \\ &\leq B + \sum_s \tilde{q}_s\lambda_s - p_{\text{gen}} \cdot \sum_s \tilde{q}_s \cdot \mathbb{E}[\tilde{M}_s(t_i)] \\ &= B - \epsilon \cdot \sum_s \tilde{q}_s.\end{aligned} \tag{2.60}$$

Application of the Foster-Lyapunov theorem completes the proof.

□

PROOF OF THEOREM 4

The proof of this theorem is closely inspired by the proof of an analogous theorem in [26]. To begin, we establish basic properties of the Dual function which follow from assumption A1.

Lemma 1. *Under assumption A1 the dual objective function $D(\boldsymbol{p})$ (2.34) is convex, lower bounded, and continuously differentiable.*

For each session $s \in S$, and price vector $\boldsymbol{p} \geq 0$, define the quantity $\beta_s(\boldsymbol{p}) : \mathbb{R}^{1+N} \mapsto \mathbb{R}^+$ as follows,

$$\beta_s(\boldsymbol{p}) := \begin{cases} \frac{1}{-f_s''\left(\lambda_s^*(\boldsymbol{p})\right)}, \text{ if } f_s'(\lambda_{\text{gen},s}^{\max}) \leq p_s \leq f_s'(\lambda_s^{\min}) \\ 0, \text{ otherwise,} \end{cases} \tag{2.61}$$

where $p_s := p_c + \sum_{u \in s} p_u$ and $\lambda_s^*(\boldsymbol{p})$ is the unique maximizer of (2.34).

For any price vector $\boldsymbol{p} \geq \mathbf{0}$ define the matrix $B(\boldsymbol{p}) = diag\big(\beta_s(\boldsymbol{p}),\ s \in S\big)$ to be the $|S| \times |S|$ matrix with diagonal elements $\beta_s(\boldsymbol{p})$. Note that from assumption A2, for all $\boldsymbol{p} \geq \mathbf{0}$,

$$0 \leq \beta_s(\boldsymbol{p}) \leq \alpha_s < \infty. \tag{2.62}$$

Define the user-session mapping matrix R to be the $N \times |S|$ matrix whose $(u,\ s)$-th entry is given by,

$$R_u^{\ s} = \begin{cases} 1, \text{ if } u \in s \text{ or equivalently } s \in S(u) \\ 0, \text{ otherwise.} \end{cases} \tag{2.63}$$

The augmented session mapping matrix $\tilde{R}$ is the $\big(1+N\big) \times |S|$ matrix whose $(r,\ s)$-th entry is,

$$\tilde{R}_r^{\ s} = \begin{cases} 1, \text{ if } r = 1 \\ R_{r-1}^{\ s},\ r \neq 1. \end{cases} \tag{2.64}$$

Lemma 2. *Under assumption A1, where it exists, the Hessian of the dual function D is given by*

$$\nabla^2 D(\boldsymbol{p}) = \tilde{R} B(\boldsymbol{p}) \tilde{R}^T. \tag{2.65}$$

Proof. Let $\nabla_p \lambda^*$ denote the $|S| \times (1+N)$ Jacobian matrix whose $(s,\ r)$-th element is $\big(\partial \lambda_s^* / \partial p_r\big)(\boldsymbol{p}),\ r \in (c,\ u\ \forall u)$. As a consequence of the Inverse Function Theorem [27] and (2.32), when it exists,

$$\frac{\partial \lambda_s^*}{\partial p_r} = \begin{cases} \frac{\tilde{R}_r^{\ s}}{f_s''\left(\lambda_s^*(\boldsymbol{p})\right)}, \text{ if } f_s'(\lambda_{\text{gen},s}^{\max}) < p_s < f_s'(\lambda_s^{\min}); \\ 0, \text{ otherwise;} \end{cases} \tag{2.66}$$

where $r \in (c,\ u\ \forall u)$. Using (2.61) we can write,

$$\nabla_p \lambda^* = -B(\boldsymbol{p}) \tilde{R}^{\mathrm{T}}. \tag{2.67}$$

From (2.36) and (2.37), $\nabla D(\boldsymbol{p}) = c - \tilde{R}\lambda$, where $c := (\lambda_{\text{EGS}}, \overline{\lambda}_u \; \forall u)$, therefore,

$$\nabla^2 D(\boldsymbol{p}) = -\tilde{R}\nabla_p \boldsymbol{\lambda} = \tilde{R}B(\boldsymbol{p})\tilde{R}^{\mathrm{T}}.$$

□

Lemma 3. *Under assumptions A1 and A2, the gradient of the dual function* $\nabla D(\boldsymbol{p})$ (2.34), (2.36), (2.37) *is Lipschitz continuous with Lipschitz constant* $L = \max_{s\in S} \beta_s(p) \cdot |S|$.

We use the following theorem to prove Lemma 3,

Theorem 6 (Rudin, 9.19 [27])**.** *Suppose* $\mathbf{f}$ *maps a convex open set* $E \subset \mathbb{R}^n$ *into* $\mathbb{R}^m$, $\mathbf{f}$ *is differentiable in* E, *and there is a real number* M *such that*

$$||\mathbf{f}'(\mathbf{x})|| \leq M,$$

for every $\mathbf{x} \in E$. *Then,*

$$|\mathbf{f}(\mathbf{b}) - \mathbf{f}(\mathbf{a})| \leq M|\mathbf{b} - \mathbf{a}|,$$

for all $\mathbf{a} \in E$, $\mathbf{b} \in E$.

Proof of Lemma 3. From Lemma 2, the Hessian of the dual function is the $\left(1+N\right) \times \left(1+N\right)$ matrix $\nabla^2 D(p) = \tilde{R}B(p)\tilde{R}^{\mathrm{T}}$. It is simple to explicitly determine the (r, r')-th entry of $\nabla^2 D(p)$. By matrix multiplication, $B(p)\tilde{R}^{\mathrm{T}}$ is the $|S| \times \left(1+N\right)$ matrix whose (s, r)-th entry is,

$$\left(B(p)\tilde{R}^{\mathrm{T}}\right)_s^{\ r} = \begin{cases} \beta_s(p), \text{ if } r = 1 \text{ or } \left(r > 1 \text{ and } s \in S(r-1)\right) \\ 0, \text{ otherwise.} \end{cases} \tag{2.68}$$

By matrix multiplication we calculate the $(r,\ r')$-th entry of $\nabla^2 D(p)$ as,

$$\begin{aligned} \left(\nabla^2 D(p)\right)_r^{\ r'} &= \sum_s R_r^{\ s} \left(B(p)R^{\mathrm{T}}\right)_s^{\ r'} \\ &= \begin{cases} \sum_s \beta_s, \ r = r' = 1 \\ \sum\limits_{s\in S(r'-1)} \beta_s, \ r = 1 \text{ and } r' > 1 \\ \sum\limits_{s\in S(r-1)} \beta_s, \ r > 1 \text{ and } r' = 1 \\ \sum\limits_{s\in S(r-1)\cap S(r'-1)} \beta_s, \ r > 1 \text{ and } r' > 1 \end{cases} \end{aligned} \tag{2.69}$$

Using the definition of the operator norm ([27], Definition 9.6 (c)) we bound the norm of the Hessian of the dual function,

$$||\nabla^2 D(p)|| \leq \max_{s\in S} \beta_s \cdot |S|. \tag{2.70}$$

The result of the lemma then follows by application of Theorem 6. Proof of Theorem 4 is assured by the following Theorem, which follows from the Descent Lemma of Convex Optimization Theory [28],

Theorem 7 ([28]). *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function and let X be a closed convex set. Assume ∇f satisfies the Lipschitz condition with Lipschitz constant L and consider the gradient projection iteration,*

$$x_{k+1} = P_x\big(x_k - \gamma \nabla f(x_k)\big),$$

with a constant step-size γ in the range $\big(0, \frac{2}{L}\big)$. Then every limit point $\overline{x}$ of the generated sequence $\{x_k\}$ satisfies the optimality condition:

$$\nabla f(\overline{x})^T (x - \overline{x}) \geq 0, \ \forall \ x \in X.$$

□

REFERENCES

[1] S. Wehner, D. Elkouss, and R. Hanson, *Quantum internet: A vision for the road ahead,* Science **362** (2018), 10.1126/science.aam9288.

[2] B. H. Charles and G. Brassard, *Quantum cryptography: Public key distribution and coin tossing,* Theoretical Computer Science **560**, 7 (2014).

[3] A. K. Ekert, *Quantum cryptography based on bell's theorem,* Phys. Rev. Lett. **67**, 661 (1991).

[4] P. Arright and L. Salvail, *Blind quantum computation,* International Journal of Quantum Information **04**, 883 (2006).

[5] A. Broadbent, J. Fitzsimons, and E. Kashefi, *Universal blind quantum computation,* in *2009 50th Annual IEEE Symposium on Foundations of Computer Science* (IEEE, Atlanta, Georgia, USA, 2009) pp. 517–526.

[6] P. Kómár, E. M. Kessler, M. Bishof, L. Jiang, A. S. Sørensen, J. Ye, and M. D. Lukin, *A quantum network of clocks,* Nature Physics **10**, 582 (2014).

[7] C. Cabrillo, J. I. Cirac, P. García-Fernández, and P. Zoller, *Creation of entangled states of distant atoms by interference,* Phys. Rev. A **59**, 1025 (1999).

[8] L.-M. Duan, M. D. Lukin, J. I. Cirac, and P. Zoller, *Long-distance quantum communication with atomic ensembles and linear optics,* Nature **414**, 413 (2001).

[9] 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 **497**, 86 (2013).

[10] 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 **558**, 268 (2018).

[11] 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 **3**, 538 (2007).

[12] 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. **130**, 050803 (2023).

[13] 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 **438**, 828 (2005).

[14] 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 **316**, 1316 (2007).

[15] 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 **607**, 69 (2022).

[16] G. Vardoyan, S. Guha, P. Nain, and D. Towsley, *On the stochastic analysis of a quantum entanglement distribution switch,* IEEE Transactions on Quantum Engineering **2**, 1 (2021).

[17] G. Avis, F. Rozpędek, and S. Wehner, *Analysis of multipartite entanglement distribution using a central quantum-network node,* (2022).

[18] L. Tassiulas and A. Ephremides, *Stability properties of constrained queuing systems and scheduling policies for maximum throughput in multihop radio networks,* IEEE Transactions on Automatic Control **37**, 1936 (1992).

[19] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, *Achieving 100% throughput in an input-queued switch,* IEEE Transactions on Communications **47**, 1260 (1999).

[20] T. Vasantam and D. Towsley, *A throughput optimal scheduling policy for a quantum switch,* in *Quantum Computing, Communication, and Simulation II,* edited by P. R. Hemmer and A. L. Migdall (SPIE, San Fransisco, California, USA, 2022) p. 1201505.

[21] R. Yehia, S. Neves, E. Diamanti, and I. Kerenidis, *Quantum city: simulation of a practical near-term metropolitan quantum network,* (2022).

[22] F. Kelly, A. Maulloo, and D. Tan, *Rate control for communication networks: Shadow prices, proportional fairness and stability,* Journal of the Operational Research Society **49** (1998).

[23] R. Srikant and L. Ying, *Communication Networks: An Optimization, Control, and Stochastic Nerworks Perspective* (Cambridge University Press, Cambridge, UK, 2014).

[24] G. Vardoyan and S. Wehner, *Quantum network utility maximization,* (2022).

[25] D. P. Bertsekas, *Nonlinear Programming, Second Edition* (Athena Scientific, Belmont, Massachusetts, USA, 1999).

[26] S. Low and D. Lapsley, *Optimization flow control. i. basic algorithm and convergence,* IEEE/ACM Transactions on Networking **7**, 861 (1999).

[27] W. Rudin, *Principles of Mathematical Analysis, Third Edition* (McGraw-Hill Inc., New York, USA, 1976).

[28] D. P. Bertsekas, *Convex Optimization Algorithms* (Athena Scientific, Belmont, Massachusetts, USA, 2015).

3

ON-DEMAND RESOURCE ALLOCATION FOR A QUANTUM NETWORK HUB

We propose an on-demand resource allocation algorithm for an Entanglement Generation Switch (EGS), where a demand is either blocked if no resources are available or else results in immediate resource allocation. We model the EGS as an Erlang loss system, with demands corresponding to sessions whose arrival is modeled as a Poisson process. To reflect the operation of a practical quantum switch, our model captures scenarios where a resource is allocated for batches of entanglement generation attempts, possibly interleaved with calibration periods for the quantum network nodes. Calibration periods are necessary to correct against drifts or jumps in the physical parameters of a quantum node that occur on a timescale that is long compared to the duration of an attempt. We then derive a formula for the demand blocking probability under three different traffic scenarios using analytical methods from applied probability and queueing theory. We prove an insensitivity theorem which guarantees that the probability a demand is blocked only depends upon the mean duration of each entanglement generation attempt and calibration period, and is not sensitive to the underlying distributions of attempt and calibration period duration. We provide numerical results to support our analysis. Our work is the first analysis of traffic characteristics at an EGS system and provides a valuable analytic tool for devising performance driven resource allocation algorithms.

3.1. INTRODUCTION

Quantum networks enable a variety of distributed applications that are not realizable via classical means alone. Among these are quantum key distribution (QKD) [1, 2], blind quantum computation (BQC) [3–5], and several entanglement-based quantum sensing techniques [6–9]. A quantum network consists of end nodes equipped with quantum hardware, as well as intermediate nodes – quantum repeaters or switches – whose main function is to enable the end nodes to carry out quantum communication tasks. In first- and second-generation quantum networks [10], these intermediate nodes use methods

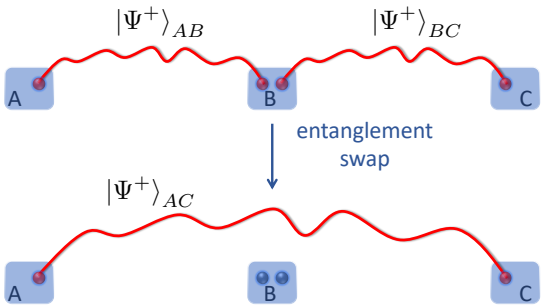


Figure 3.1: A simple quantum network with end nodes A and C wishing to share entanglement, and an intermediate node B assisting them with the task. Initially, two entangled links – $|\Psi^+\rangle_{AB}$ between A and B and $|\Psi^+\rangle_{BC}$ between B and C – are established. B then performs a swapping operation to directly entangle A and C's qubits. Depending on the distance between A and C, direct generation of entanglement (without an intermediate node), may not be feasible.

such as entanglement swapping as shown in Figure 3.1 to provide entanglement as a resource to user-run applications – either to be consumed directly, or used in teleportation-based transport of quantum information [11, 12]. When multiple applications have simultaneous demand for shared and limited resources (e.g., quantum memories, links, measurement modules), contention may arise, and the network must enact a resource allocation scheme.

In this work, we study a type of quantum network hub previously referred to as an Entanglement Generation Switch (EGS) [13]. An EGS is a type of quantum switch with control over a pool of resources which, when allocated to a set of nodes, enable a task such as entanglement generation. Unlike its memory-equipped counterpart (sometimes referred to as an entanglement distribution switch, or EDS), the EGS is relatively easy to fabricate since it has no memories: it possesses a number of resources such as Bell state analyzers (BSAs) [14–16], which serve as a means of performing probabilistic optical entanglement swapping on incoming photons (each entangled with a qubit at an end node), and upon a successful swap generating end-to-end entanglement. This method of generating entanglement is illustrated in Figure 3.2, and the process is explained in more detail in Section 3.3. In principle, an EGS can serve any number of nodes with a single shared BSA, but more BSAs can ameliorate contention for this resource; see Figure 3.3 for an example of an EGS with three BSAs to service four nodes. Some EDS proposals on the other hand additionally place BSAs in the middle of each physical link that connects the device to other nodes in the network. While these BSAs assist with entanglement generation at the link level, the resulting architecture is fairly demanding in the number and type of hardware components: K links translate into K dedicated BSAs and at least an equal number of quantum memories at the EDS. In contrast, the EGS architecture places all BSAs at the central hub along with a switching fabric for reconfiguration so as to serve any set of end nodes, resource limitations permitting. An enabling technology for an EGS is an optical switch linking N inputs to a pool of C resources (e.g., BSAs), such that any

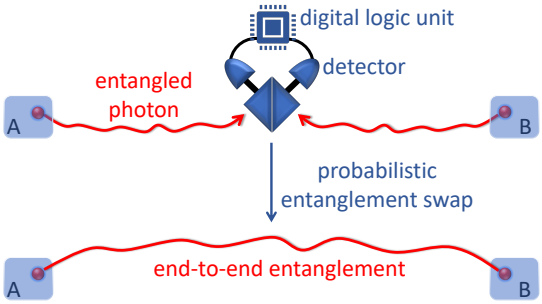


Figure 3.2: Entanglement generation with a Bell state analyzer. Entangled photons travel towards the BSA, which carries out a probabilistic entanglement swap. At the BSA, photons pass through a beam splitter whose output ports are connected to a pair of detectors. The digital logic unit reads out measurement results and determines if a swap was successful. Results are classically communicated to nodes A and B, whose qubits become entangled upon a successful event.

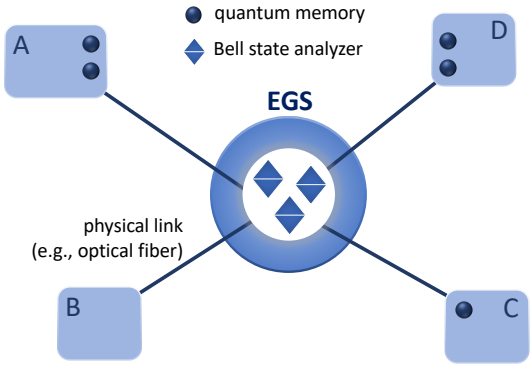


Figure 3.3: End nodes being serviced by an EGS with a pool of three BSAs.

pair of inputs can be allocated any resource. Architectures for such switches exist, see e.g., [17] for the presentation and evaluation of optimal switching architectures based on 2×2 optical switches, or [18] for foundational work on re-arrangeable non-blocking switches.

With these properties, the EGS is poised to be an excellent candidate for a scalable and straightforwardly implementable quantum network component, especially in the Noisy Intermediate Scale Quantum (NISQ) era [19]. While the EGS can be used to directly connect end nodes, as shown in Figure 3.3, it can also provide entanglement to other intermediate quantum network nodes, e.g., quantum repeaters/switches equipped with quantum memories of sufficiently long coherence time, each servicing a quantum local area network, as shown in Figure 3.4. The versatility of the EGS warrants investigation into its practical operation within a quantum network; we provide a detailed explanation

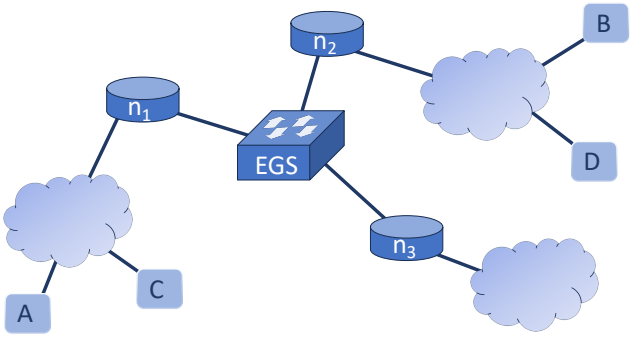


Figure 3.4: Intermediate quantum network nodes leveraging the EGS to interconnect their respective quantum local area networks.

of this in Section 3.3.

We model the EGS as a fixed-capacity facility operating as an Erlang loss system. This is a classic queueing model (see e.g., [20, 21]) where arriving sessions are immediately rejected (not queued) when all resources are busy. This blocking behavior minimizes the response time with which service decisions are communicated, as queueing sessions naturally introduces latency before a session is served. A trade-off in performance arises, in that blocked sessions do not receive service, whereas queued sessions may typically receive service, albeit following an arbitrary delay. Response time is a critical performance metric for on-line entanglement routing protocols [22]. For example, in sequential entanglement swapping schemes [23, 24] end nodes must store entanglement while the end-to-end path between sender and receiver is constructed hop-by-hop, making the process highly sensitive to the latency of routing decisions [22]. Such routing based use cases for the on-demand resource allocation schemes we propose are particularly compatible with EGS systems operating as interconnects between several local areas, as illustrated in Figure 3.4.

We assume that entanglement requests arrive at the EGS as sessions according to a Poisson process, where each session consists of multiple entanglement generation attempts. We then analyze three operational modes of the device, while heeding physical capabilities and limitations of both the EGS and the nodes connected to it. Namely, inspired by realistic expectations of hardware characteristics in the near term, we equip the system with two important and pragmatic features: (i) batching of entanglement generation attempts due to generally high rates of failure; and (ii) provisioning for calibration periods necessitated by the quantum communication qubits of nodes that are served by the EGS. In the first operational mode scenario we model strict resource reservation, where an accepted demand is scheduled for several batches of entanglement generation interleaved with calibration periods and does not release the resource until all attempts are complete or entanglement generation succeeds, whatever happens first. The second scenario we study is a variation of the first in that a successful attempt does not trigger

release of the EGS resource; all attempts are carried out, thus possibly resulting in multiple successfully-established entangled states. In the final scenario, nodes relinquish EGS resources during calibration periods, and must re-obtain them afterwards – we opt for jump-over blocking to describe the retrial behavior of this system. In all scenarios, an incoming request is blocked (dropped) if upon arrival it sees no free resource at the switch. While studying the EGS in the context of these different operation modes, we make the following contributions:

- We provide a comprehensive description of EGS operational details, with system specifications rooted in practical considerations of the underlying physical architecture;
- We model the EGS as an Erlang loss system with sessions of demands arriving as a Poisson process, which we subsequently analyze to obtain (i) the stationary distribution of the number of active requests being served at the switch; (ii) request blocking probabilities; and (iii) an insensitivity result that highlights the broad applicability of our model to practical systems; We highlight that in the third EGS operation mode (with resource relinquishment and retrials), the blocking probability is identical for the first attempt of a session as well as all retrials within that session. This property simplifies application of the result, as only one formula needs to be evaluated.
- We develop an extensive simulation framework capable of enacting sequences of events that model the operation of a real EGS – one that operates in discrete time – in a variety of configurations. Simulation code will be made available to enable future studies of the EGS.

The numerical results obtained from our continuous-time switch model closely match the more realistic discrete-time simulation of the switch. The physical relevance of our model, both in the hardware design we consider, as well as system control protocols we propose, set this work apart from much of the previous literature, wherein hardware limitations are frequently understated. The wide scope of our framework moreover enables one to model arbitrary traffic patterns and a wide variety of hardware settings, including ones where nodes have multiple communication qubits.

The rest of this manuscript is organized as follows: in Section 3.2, we provide the relevant queueing-theoretic and quantum switching background. In Section 3.3, we outline the system description, including physically-motivated operation settings. In Section 3.4, we introduce the model of the EGS and state our assumptions. Section 3.5 presents the analysis, while Section 3.6 provides a numerical evaluation of the system. We make concluding remarks in Section 3.7.

3.2. BACKGROUND

An EGS serves a role analogous to a central hub in a system of telephone lines, managing the connection of pairs of quantum nodes to resources, much like a telephone exchange directs and facilitates communication between sets of callers. Traditionally, telephone systems are studied using the Erlang loss model, wherein calls arrive according to a Poisson process to a server with a total of C telephone lines. An incoming call will be blocked if all C lines are occupied upon arrival. The blocking probability of a call is computed using the well-known Erlang formula [20]. It has been shown that this model

exhibits insensitivity to the type of service time distribution of calls, as the blocking formula depends only on the *average* service time of calls [25]. This result follows from the argument that the underlying Markov process describing the system is a partially reversible one, which is a necessary and sufficient condition to have insensitivity [26]. A common method to prove insensitivity is to first study the given queueing model assuming service time distributions are Coxian (these are dense in the class of distributions with nonnegative support [27]), and then show that the stationary distribution or blocking formula is the same as in the case of an exponential service time distribution with the same mean. Then, by using the continuity of queueing models to service time distributions [28], the insensitivity property also holds for generally distributed service times, which can be approximated by Coxian distributions to an arbitrary degree of accuracy. The insensitivity property is a useful tool to dimension a practical system with a general service time distribution by studying the same system with the simpler case of an exponential service time distribution with the same mean.

In [29], Bonald studied the scenario where requests are generated as sessions that arrive according to a Poisson process, with each session containing several calls. It was shown that even in this case, the Erlang model is insensitive to service time distributions. In our model, calls also arrive as sessions, with each session consisting of several attempts for entanglement generation to describe practical quantum systems where entanglement requests arrive in batches from an application. We also assume sessions arrive according to a Poisson process, which will be a valid assumption when a large number of users or applications trigger entanglement requests. The analysis of this paper is based on the analysis of [29], albeit our aim here is to analyze a quantum system which has distinctive features when compared to classical systems. Our analysis is significantly different from [29] due to the presence of new parameters and characteristics of quantum systems, and subsequently analytical expressions for blocking probabilities for our model are different from those presented in [29]. We show that a quantum switch that can be modelled as an Erlang model also exhibits insensitivity to service time distributions under certain necessary conditions.

The EGS architecture was initially introduced in [13], where the authors highlighted the scalable properties of this type of hub. The authors then proposed and studied a Rate Control Protocol whose aim is to modulate user demand rates to the switch based on the EGS's capacity to serve users, as well as on overall traffic trends. The focus of this work is mainly on fair resource allocation, achieved through a network utility maximization-based [30] framework. In contrast, the protocols proposed in our work use request blocking instead of rate control as means of resource management. Furthermore, our work aims to accurately represent the EGS in a discrete setting, with concrete descriptions of request structure and procedures for request handling.

Memory-equipped quantum switches (EDSs) have been extensively investigated from queueing-theoretic and request scheduling perspectives, see e.g., [31–34]. In [32] the authors modeled an idealized EDS as a discrete-time Markov chain and computed its capacity under a simple swapping protocol. While this study shares similarities with ours in terms of the focus on discrete-time system evolution, important differences exist. As briefly outlined in the previous section, EDS-based models often assume the presence of quantum memories both at the end nodes and at the switch ([32], for instance assumes

infinitely-sized buffers). Each node may even have dedicated infrastructure connecting it to the hub, including a pre-allocated quantum switch buffer and midpoint stations located at each link to herald entanglement. This enables switch-node entanglement to be attempted independently by each link, and successfully generated states to be stored until a swapping opportunity arises. With such an abundance of resources, contention is minimal, and resource allocation is not a primary focus. In another line of work on EDS-based models, contention among entanglement requests of different flows for using switch-node entanglement was studied in [34, 35]. In [35], a throughput optimal scheduling policy was designed for a quantum switch model when switch-node entanglement has a short lifetime of one time slot. They proved that the queues of entanglement requests are stable under their max-weight scheduling policy for all feasible entanglement request rates. A similar model with the scenario where switch-node entanglement has infinite lifetime was studied in [34].

In contrast to the EDS models, the EGS lacks memories, necessitating resource solicitation by nodes, followed by entanglement generation attempts executed in a synchronized manner to ensure nearly simultaneous photon arrival at the hub. Furthermore, our EGS protocols involve batched attempts interleaved with periods of EGS inactivity, effectively constituting extended "sessions" of engagement with EGS resources. The system studied in our work thus exhibits both architectural and algorithmic differences to the EDS, requiring novel and tailored analytical methodologies.

Quantum switches, both of the EGS and EDS types may be compatible with functional integration with quantum repeaters. Indeed, as with the EGS implementation we describe, several repeater architectures (see e.g., [36–40]) rely on the interference and measurement of photons by a BSA located at a midpoint between other nodes of the network. Quantum switches differ physically and operationally from repeaters because they require a switching fabric and protocols to effectively mediate contention for shared resources, whereas quantum repeaters are not subject to these requirements.

3.3. SYSTEM DESCRIPTION

An EGS consists of three main components: (1) A pool of allocatable *resources* such as BSAs; (2) A *switch* capable of allocating any resource to any pair of nodes connected to the EGS; (3) and a *processor* capable of scheduling the allocation of resources to pairs of nodes, controlling the operation of the switch, and sending and receiving classical messages. An EGS with control of a pool of three BSA-type resources is illustrated in Figure 3.3. Nodes are connected to an EGS by physical links, such as optical fiber connection. To gain access to an EGS resource, pairs of nodes send a message to the EGS called a *request*.

Definition 9. *Request. Nodes n_i and n_j who wish to communicate issue a* request *to the EGS. A request is a demand for the generation of one or more EPR pairs between n_i and n_j. The exact composition of a request depends both on the nodes' physical capabilities as well as on application-focused goals.*

For every distinct pair of nodes (n_i, n_j), one node is designated the *initiator* and the other the *secondary node*. Requests are communicated to the EGS by the initiator node. A node consists of the following set of components: (i) one or more *communication*

qubits, each capable of preparing a quantum state and emitting one or more photons[1]; (*ii*) devices needed to manipulate the state(s) of the communication qubit(s) – examples include lasers, waveform generators and microwave sources; (*iii*) devices needed to measure a communication qubit; (*iv*) possibly one or more quantum memories to which the quantum state of a communication qubit may be swapped, capable of storing the state for a finite period of time; (*v*) and a classical processor capable of controlling the states prepared in communication qubits, triggering swaps to memory, triggering measurement of a communication qubit, and sending and receiving classical messages.

In Figure 3.3, the nodes connected to the EGS are assumed to be limited in that they either have few or no quantum memories, and may have a restricted number of communication qubits. Figure 3.4 illustrates a scenario where the EGS may be connected to more powerful nodes, potentially with access to many communication qubits and quantum memories per node. As in Figure 3.4, nodes can function as relays or intermediates interfacing with several quantum local area networks.

3.3.1. Physical Operation Settings

Bipartite Heralded Entanglement (HE) generation and generation of Correlated Information (CI) are two ways in which a pair of nodes can interact via the EGS. In Appendix 3.8.5, we describe a protocol for the latter, while here we describe in detail a protocol for bipartite HE generation [36, 37] that is based on a single-click scheme [42]. This method of producing entanglement has been successfully demonstrated in several experimental platforms, including Color Centers [42, 43], Ion Traps [44, 45], Atomic Ensembles [46, 47] and Neutral Atoms [48]. Applications of HE generation include BQC, teleportation and clock-synchronization [7], and an application of CI generation is Measurement Device Independent QKD (MDI-QKD) [49, 50]. Each of these tasks can be enabled by an EGS where the shareable resource is a BSA. The EGS can be equipped with multiple BSAs (Figure 3.3), or more generally with a different type of resource, in order to support other interaction protocols between quantum network nodes. Each of the protocols we describe is compatible with a BSA that consists of a 50/50 beam splitter with two input channels; each of the two output ports of the 50/50 beam splitter is connected to a photon detector; the outputs of the photon detectors are connected to a digital logic unit such as a Field Programmable Gate Array (FPGA) which processes the measurement outcomes and can communicate a success/failure flag back to the nodes of a flow. Such a BSA is depicted in Figure 3.2. In Table 3.3 (Section 3.6) we provide an inventory of physical and protocol parameters motivated by implementations of single click bipartite HE generation based on experimental demonstrations with a Nitrogen Vacancy (NV) center in diamond [42, 51].

Heralded entanglement generation

The goal of node pair (n_i, n_j) running the bipartite HE generation protocol is to entangle a communication qubit of node n_i with a communication qubit of node n_j. The term "bipartite" thus refers to the property of the protocol that the resulting entanglement involves two qubits. We will sometimes refer to such states as Einstein–Podolsky–Rosen (EPR) pairs or Bell states. The protocol is called *heralded* because for every attempt to

[1] In some simple realizations of a node, the communication qubits may be replaced by all photonic state preparation devices. See [41] for an example.

generate entanglement that the nodes make, a success or failure flag is generated and converted into a message that is sent to the nodes, thereby indicating if the attempt succeeded or failed. Triggering a subsequent attempt after a success would destroy the entanglement that was created. To prevent wasting entanglement, the protocol includes a wait time for the heralding flag to arrive before triggering subsequent attempts. In certain EGS operation modes an attempt may not be triggered if a success flag is received (see Section 3.3.1). In a setting where node pairs use the entanglement generated between them to perform some application, this protocol is beneficial because it allows the nodes to condition commencement of the application on the successful generation of entanglement.

At a high level, a single-click heralded entanglement generation protocol consists of four stages. First, each node performs a sequence of calibration operations and prepares a communication qubit in a known state. Second, each node locally triggers the generation of entanglement between the state of their communication qubit and the presence/absence of a travelling photon. Third, the presence/absence encoded photons are sent to a BSA, at which a Bell-State Measurement (BSM) (entanglement swap) is attempted between the encoded photons. Fourth, if the BSM succeeds the communication qubits of the two quantum processing nodes will have become entangled and a success flag is sent to the nodes. If the measurement is not successful a failure flag is communicated to the nodes. The second, third and fourth stages occurring sequentially constitute a single HE generation attempt. For the example physical platform of the NV center in diamond, the calibration operations correspond to a Charge and Resonance (CR) check [51].

Attempts can be repeated in batches of $N_{\text{attempts in batch}}$ sequential attempts that are interleaved with repetition of the first step – calibration of the communication qubit – following the last attempt in a batch. The main limitation on the batched attempt repetition rate is the Round Trip Time (RTT) of communication associated with the third and fourth stages of an attempt. The need to wait for the arrival of the heralding flag especially limits the rate, yet this is necessary to prevent the destruction of created entanglement by triggering a new attempt. This aspect of the protocol motivates an assumption in our mathematical models that entanglement generation attempts are non-overlapping. For any system where an individual attempt to generate entanglement has a low probability of success, it is beneficial to allow such batching of attempts, which increases the probability a success will occur within any finite amount of time.

A communication sequence between node pair (n_i, n_j), which requests and is allocated use of an EGS resource to perform HE generation, is included in Appendix 3.8.5 in Figure 3.18b.

PROBABILITY OF SUCCESS

We model experimental implementations of HE generation where the state of the communication qubit is reset (stage one) at the start of each attempt and every attempt in a batch corresponds to an identical experimental sequence. Furthermore we assume that the characteristics of devices used in triggering entanglement generation attempts, such as laser pump power and frequency, remain constant. Similarly, we assume that the optical characteristics of the path from the communication qubit to an allocated resource, including any loss introduced by the switch at the EGS, remain constant. Therefore, the

probability of entanglement generation may only change over attempts if there are physical parameters that drift or jump over a batch of attempts. For any system where attempts have a fixed mean duration that is short in comparison to the parameter drift/jump timescales, such effects may be accounted for by assuming that the probability of successful entanglement generation is a function of the jth attempt in a batch of attempts, $p_{\text{gen}}(j)$.

For an implementation in the NV colour center in diamond, one may assume that the outcomes of sequential attempts in a batch are identically and independently distributed (IID), with a fixed probability of success p_{gen} [42]. This assumption is valid as long as calibration periods are performed frequently enough between batches of attempts to prevent slow effects – such as the spectral diffusion which affects solid state quantum emitters – from corrupting the state of the communication qubit. The assumption that the outcomes of sequential attempts are IID with a fixed probability of success p_{gen} also applies to other experimental platforms, such as Trapped Ions [44, 45], where the mean attempt duration is significantly shorter than sources of parameter drift. The assumption that p_{gen} is constant and is independent of attempt duration distributions but depends only on the mean attempt duration is the necessary condition that we use in proving the insensitivity result discussed in the introduction section (this is Theorem 9 in Section 3.5).

SINGLE VS MULTIPLE ENTANGLEMENT GENERATION

For a limited quantum node, such as a node with one communication qubit and possibly a memory, it may be most practical to engage in single entanglement generation. That is, if an attempt to generate entanglement succeeds, no further attempts in a batch will be executed. Physically, successful entanglement generation renders the communication qubit of the device unavailable for further attempts until that entanglement can be used or transferred to memory. Transfers to memory are not instantaneous and have a finite time cost, thus communication qubits can not be freed instantly even in a system with memory. Moreover, if a communication qubit is coupled to a memory, attempts to generate entanglement while a state is stored in memory may damage the stored state due to induced decoherence [51]. This effect results from a persistent non-zero coupling to the memory. Single entanglement generation may be the preferential operation mode of limited quantum devices to account for these effects [51]. In contrast, a quantum node with multiple communication qubits may leverage them to generate multiple entangled states, possibly by multiplexing photon emission from the node.

3.4. MODEL AND ASSUMPTIONS

In this section, we lay the groundwork for the analysis of a star-topology system with the EGS at its center, as shown in Figure 3.3. To this end, we first state the model and introduce the bulk of our notation. We then present several abstractly defined terms for the periods that comprise a session (a request for EGS resource access). These terms bridge the gap between the physical and queueing-theoretic models of the EGS. We then discuss various EGS operation modes, which we call service models. A service model describes how the EGS handles requests, including:

(1) resource reservation, specifying the amount and duration of resource allocation to a

pair of communicating nodes;

(2) *retrial behavior*, specifying actions taken upon blocked service events; and

(3) *termination behavior*, specifying events that trigger the EGS to end service to a pair of nodes.

Modeling assumptions are introduced throughout, and we conclude with a table of notation.

3.4.1. SYSTEM MODEL AND NOTATION

NETWORK COMPONENTS

The network consists of a star-topology system with the EGS at its center, in which the EGS serves K quantum nodes. Node n_i, $\forall i \in \{1, \cdots, K\}$ has a total of c_i communication qubits which can be used for entanglement generation, while the EGS has C resource modules (e.g., Bell State Analyzers).

SESSIONS

A request issued from nodes n_i, n_j to access an EGS resource triggers the creation of a *session.*

Definition 10. *Session. A* session *between nodes n_i and n_j, denoted by the tuple (n_i, n_j, t) is a specification of the batches of entanglement generation attempts that are required to satisfy a request of type t. The specification includes a number of attempts per batch and a number of batches requested. The batches may be interleaved with idle and/or calibration periods. A session makes use of a single communication qubit at each of nodes n_i, n_j.*

The definition above makes a reference to a *session type*: as in [29], we permit the existence of differently-structured sessions within one system. Physically, these may encode the technical capabilities of nodes, such as the maximum number of attempts in a batch before calibration operations need to be repeated. Alternatively, session types may correspond to different applications, entanglement distribution algorithms, or even application *instantiations.* An example of the latter is a pair of nodes (n_1, n_2) that assist in carrying out two QKD instantiations across the network shown in Figure 3.4: one for the user pair (A, B) for a requested key size M_1, and the other for the user pair (C, D) for a requested key size $M_2 > M_1$. The session type corresponding to C and D's request would require more entanglement generation attempts over the course of all batches, and possibly more calibration or idle periods, depending on physical system restrictions or the algorithmic design of the EGS control protocol, respectively. Another example is the use of session types to accommodate application-dependent fidelity requirements: an application with a high minimum fidelity threshold can for instance request a session with more attempts in the hopes of producing enough states from which to distill higher-fidelity entanglement [52, 53].

FLOWS

The distinct possibilities for the sources of session requests are known as flows. A flow is associated with a pair of nodes, (n_i, n_j) and a session type t. The set $\mathscr{F}$ of all possible

flows is

$$\mathscr{F} \equiv \{f_{i,j}^{t} : i, j \in \{1,\ldots,K\}, t \in \{1,\ldots,T\}\}, \tag{3.1}$$

where K is the number of nodes connected to the EGS and T is the total number of possible session types. The cardinality of $\mathscr{F}$ is $F = \binom{K}{2} \times T$, since each node-pair could run any of the T session types.[2] To streamline notation, we denote flows based on their index in $\mathscr{F}$, so the k^{th} flow is denoted by f_k, $k \in \{1,\cdots,F\}$. We note that while flows uniquely identify a type of session between a pair of nodes, sessions need not be unique: n_i and n_j can have multiple concurrent sessions of type t, as long as resources (communication qubits and EGS resources) are available. We introduce the notation $n_k \in f$ to mean that node n_k partakes in flow f.

Session Request Arrivals

Requests for a new session by flow f are triggered according to a Poisson process with rate ν_1^f, where the sub-script indicates arrival to the first period of the session (see Appendix 3.8.1 for greater detail). However, each request is only actually submitted to the EGS if both participating nodes have an available communication qubit with which to execute a resulting session (one of total c_k communication qubits at node n_k). No request is submitted if one or both nodes does not have an available communication qubit. Once admitted, a session occupies one communication qubit from each node for its entire duration. We assume that individual sessions for a flow are independent.

System State and Request Blocking

The system state is characterized by the number of active sessions for each flow:

$$\mathbf{q} = [q_1,\ldots,q_F], \tag{3.2}$$

where $q_j \in \mathbb{Z}_+$ is the number of ongoing sessions of flow f_j, and $\mathbb{Z}_+$ indicates the set of non-negative integers. A request to reserve an EGS resource is blocked (rejected) if admitting it would violate the EGS capacity constraint (C resources).

Traffic Intensity

For every flow $f \in \mathscr{F}$, we define a quantity known as the *traffic intensity*,

$$\rho^f = \nu_1^f \times \mathbb{E}[\text{session duration}], \tag{3.3}$$

where the session duration includes all periods in which a communication qubit of each node in the flow remains occupied (details to follow in Section 3.4.2). The traffic intensity is a quantity which often arises in queuing theory. It is more typically defined as the ratio of the arrival rate of a session to the service rate of a session. As the mean session duration is equal to the inverse of the mean session service rate, (3.3) is equivalent to the standard definition. In Section 3.5 we will show that the session blocking probabilities only depend on these flow-level traffic intensities, rather than on the detailed distributions of session components.

[2] It is possible that certain node-pairs do not have the physical capability to carry out sessions of a given type t. The state space can be easily amended to reflect such restrictions; we make the assumption that all node-pairs are capable of all session types merely to simplify notation.

3.4.2. SESSION STRUCTURE

A session consists of a sequence of periods that differ in their resource usage patterns. The types of periods in a session form the basis of the service models, developed in Section 3.4.3.

PERIOD TYPES

Definition 11. *Attempt. An* attempt *is the basic service component for two nodes communicating via the EGS. A attempt involves the active use of an EGS resource to make one entanglement generation attempt. An attempt consumes one EGS resource and one communication qubit from each node.*

An individual attempt period for flow f is denoted $\mathcal{A}_j^f$, where $j \in \{1, \cdots, M_A^f\}$. For the purpose of this work, we establish two additional service component types which are not attempts in that EGS resources are not in active use for their duration. For the first of these, we follow the convention set by [29] to define *idle periods*.

Definition 12. *Idle period. When nodes n_i and n_j enter an* idle period*, they relinquish all EGS resources, so that a subsequent call would require a new service reservation. Neither EGS resources nor active qubit operations are required. However, one communication qubit of each node remains reserved (unavailable for new session requests) during an idle period.*

We leave relaxations of the assumption that idle periods engage one communication qubit of each node for future work, as this assumption simplifies the analysis.

In our EGS model, there is also a possibility that of two communicating nodes n_i and n_j, one or more necessitates a calibration period after a number of active (attempt) periods. The duration of such calibration periods typically has a finite mean, albeit it can be randomly distributed. Depending on the service model, the nodes may decide to not relinquish resources they have already reserved at the EGS.

Definition 13. *Calibration period. A* calibration period *between nodes n_i and n_j requires node hardware resources from one or both nodes and engages one communication qubit of each node. No entanglement generation attempts occur in a calibration period. Depending on the service model, however, the nodes may continue to hold onto EGS resources for the duration of a calibration period, precluding other nodes from accessing them.*

We assume in this work that a calibration period engages one communication qubit of each node, as opposed to, e.g., all qubits of a node. While the latter scenario may also be of interest, it poses a challenge for analysis since service requests can no longer be treated independently from each other. We also remark that if the nodes relinquish EGS resources at the beginning of a calibration period, then from the perspective of the EGS (and from a modeling perspective) the period is an idle one. We distinguish between calibration and idle period types because calibration periods are always physically motivated: they necessarily engage quantum hardware at nodes. In contrast, idle periods need not stem from quantum hardware restrictions at nodes (even if we assume that communication qubits are unavailable for new service request creation during idle periods). Examples of these are an entanglement generation attempt followed by a *classical* processing period at the nodes, or a link-layer protocol that imposes a back-off timer between successive entanglement generation attempts.

SESSION COMPOSITION

A session of flow type f consists of:

- M_A^f attempt periods, with mean duration $1/\eta_{A,j}^f$, $\forall\, j \in \{1, \cdots, M_A^f\}$.
- M_C^f calibration periods, each with mean duration $1/\eta_{C,j}^f$, $\forall\, j \in \{1, \cdots, M_C^f\}$.
- M_I^f idle periods, each with mean duration $1/\eta_{I,j}^f$, $\forall\, j \in \{1, \cdots, M_I^f\}$.

These periods may be interleaved in various patterns depending on the service model. By convention, sessions always begin and end with attempt periods rather than calibration or idle periods.

We model periods by Coxian phase-type distributions, in which a single period is decomposed into a sequence of N phases, each with exponentially distributed duration such that the mean period duration is preserved. As discussed in Section 3.2, these distributions can approximate generally distributed periods. The accuracy of the approximation can be made arbitrarily accurate by increasing N. It is physically relevant to model periods in this way, to ensure that our results can be applied to real systems, in which periods can follow general distributions. In Appendix 3.8.1 we discuss our model at the phase-level of detail. The key results of our analysis, presented in Section 3.5 only include quantities at the flow-level of detail.

MEAN SESSION DURATION

The duration of a session of flow type f depends on the service model and the mean duration of each period. In the physical systems our model describes, each type of period may have a generally distributed duration. However, we show in Section 3.5, Theorem 9, that the session blocking probabilities are *insensitive* to these distributions. As a result, only the mean durations of periods contribute to the blocking probability, which happens via their contribution to the traffic intensity of flow f, ρ^f.

For example, if a session comprises M_A^f attempt periods each with average duration $1/\eta_A^f$, interleaved by M_C^f calibration periods each with average duration $1/\eta_C^f$ and no periods are skipped, then the mean session duration is

$$\mathbb{E}[\text{session duration}] = \frac{M_A^f}{\eta_A^f} + \frac{M_C^f}{\eta_C^f}.$$

However, some service models allow early termination (see Section 3.4.3).

3.4.3. SERVICE MODELS

We study three distinct service models that differ in their resource reservation policies, retrial behavior, and termination conditions. The key differences between service models are summarized in Table 3.1.

Feature	Single EPR Pair Generation with Strict Resource Reservation	Multiple EPR Pair Generation with Strict Resource Reservation	Multiple EPR Pair Generation with Resource Relinquishment
Idle periods	None	None	Present
EGS resource status during node calibration	Held by session	Held by session	Relinquished by session (calibration takes place in idle periods)
Retrial mechanism	None	None	Jump-over blocking
Session termination trigger	First successful attempt OR all attempts complete	All attempts complete	All retrials complete AND all attempts complete in accepted retrials

Table 3.1: Summary of the three different service models based on their resource reservation, retrial, and termination behaviors.

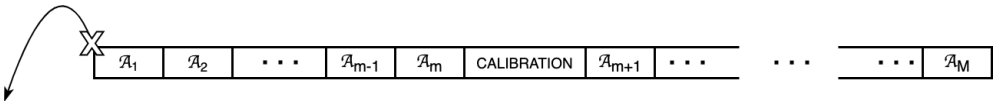


Figure 3.5: Strict resource reservation service model. A session consists of multiple EPR pair generation attempts, denoted by $\mathcal{A}_i$, $i = 1,\ldots,M_A$. A calibration period is carried out after every m attempts. In the "multiple EPR pair generation" variant of this service model, an admitted session does not relinquish resources for its entire duration, while in the "single EPR pair generation" variant, the session ends after a successful attempt.

SINGLE EPR PAIR GENERATION WITH STRICT RESOURCE RESERVATION

In this service model a session consists of batches of attempt periods interleaved with calibration periods. Once a session is admitted at the EGS it holds onto its EGS resource until termination, even during calibration periods. The session terminates either when:

- An entanglement generation attempt succeeds, or
- All M_A^f attempts have been completed.

There are no idle periods in this service model, so $M_I^f = 0$. A calibration period is performed after every m attempts to maintain hardware performance. The strict resource reservation session structure corresponding to this service model is illustrated in Figure 3.5.

In this service model, the mean duration of a session for flow type f is given by

$$\mathbb{E}[\text{session duration}] = \sum_{j=1}^{M_A^f} \frac{\mathscr{P}_{A,j}^f}{\eta_{A,j}^f} + \sum_{j=1}^{M_C^f} \frac{\mathscr{P}_{C,j}^f}{\eta_{C,j}^f}, \tag{3.4}$$

where $\mathscr{P}_{A,j}^f$ and $\mathscr{P}_{C,j}^f$ are the probabilities of reaching the j^{th} attempt and calibration periods, respectively. These probabilities account for the possibility of early termination of the session due to a successful entanglement generation attempt.

MULTIPLE EPR PAIR GENERATION WITH STRICT RESOURCE RESERVATION

This service model is identical to the previous service model, with the exception that a successful entanglement generation attempt does not trigger early termination. Instead, all M_A^f attempts are carried out, potentially producing multiple EPR pairs. The session terminates only when all attempts are complete. As this service model also uses strict resource reservation, the related session structure is again illustrated by Figure 3.5.

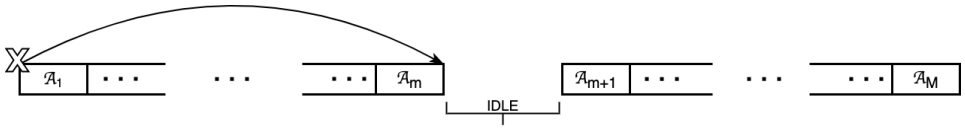


Figure 3.6: Service model with resource relinquishment and jump-over blocking. A session consists of multiple "active" (periods which make use of an EGS resource module), denoted by $\mathcal{A}_i$, $i = 1,\ldots,M_A$, interspersed with idle periods. In jump-over blocking, a blocked session goes to the beginning of the next idle period, if there is one, or terminates in case no idle periods remain in the session.

Since early termination cannot occur, the mean session duration simplifies to

$$\mathbb{E}[\text{session duration}] = \sum_{j=1}^{M_A^f} \frac{1}{\eta_{A,j}^f} + \sum_{j=1}^{M_C^f} \frac{1}{\eta_{C,j}^f}. \tag{3.5}$$

In Section 3.3 we discussed physical considerations pertaining to single vs. multiple entanglement generation operation modes. These considerations imply that there is some subtlety as to what happens physically when entanglement is successfully generated in this service model, as well as in the following service model. If an attempt succeeds, the node will either proceed to consume that entanglement directly for some application or swap the entanglement to memory. Either of these options have a finite time cost. However, in this system model we assume that even if an attempt succeeds the communication qubit of each node associated with the session is available by the start of the next attempt, so that it may be executed. This assumption is an idealization which applies in the limit where the consumption of entanglement or swapping to memory can be completed instantaneously. Practically however, this model is still of interest for nodes which consume entanglement or execute swaps to memory on reasonably short time scales, as compared to the duration of a batch of attempts. In an operational setting such a node may simply skip a small number of subsequent attempts, until these tasks are concluded. Such an action will slightly reduce the expectation value of the number of entangled pairs that the session will produce, but will have no impact on the blocking probability as long as the nodes do not relinquish their EGS resource.

Multiple EPR Pair Generation with Resource Relinquishment

In this service model, sessions consist of *active periods*, which make use of EGS resources, interleaved with idle periods, during which resources are relinquished. At the end of each idle period, the session attempts to re-obtain an EGS resource module by submitting a request. The request can be blocked by the EGS if all C resources are occupied. Failure to obtain a resource (either at the beginning of a session or after an idle period) triggers a *jump-over retrial*: the session either transitions to the next idle period, or if one does not exist, terminates. A distinct feature of jump-over retrials is that when a retrial is blocked, the corresponding batch of attempts is skipped entirely. Hence the number of attempts that are actually conducted in a session is not fixed, but depends on the success of retrials. Notably, if the initial request and all retrials of a session are blocked, then a session may terminate from the final idle period without a single entanglement generation attempt

having taken place. The structure of sessions corresponding to this service model is illustrated in Figure 3.6.

As in the other service models, one communication qubit of each node remains reserved throughout the session duration, including during idle periods and retrials. In contrast to the other service models, EGS resources are held only during active periods.

In this service model, the mean session duration depends non-trivially on the request blocking probabilities for the first request of the session, as well as for each subsequent retrial. In Appendix 3.8.4 we prove that the request blocking probability $\overline{\pi}_f$ for a session of flow type f does not depend on whether it is the first request of the session or a retrial. The mean session duration is given by

$$\mathbb{E}[\text{session duration}] = \sum_{j=1}^{M_A^f} \frac{\mathscr{P}_{A,j}^f}{\eta_{A,j}^f} + \sum_{j=1}^{M_I^f} \frac{1}{\eta_{I,j}^f}, \tag{3.6}$$

where the probability $\mathscr{P}_{A,j}^f$ of reaching the j^{th} attempt depends on the blocking probability $\overline{\pi}_f$ and how the attempts are partitioned into batches by the session structure.

For reference, Table 3.2 provides an overview of the notation used throughout our analysis.

3.5. ANALYSIS

In this section, we present the results of our analysis of the three EGS service models: single and multiple EPR pair generation with strict resource reservation, and multiple EPR pair generation with resource relinquishment during idle periods. The main quantities of interest are closed-form expressions for the probability that an arriving request is blocked. This is an event that occurs when said request sees all C EGS resources engaged. In strict resource reservation models, blocking can only occur at the beginning of a session. On the other hand, if sessions contain idle periods, as in the resource relinquishing model, then blocking may also occur throughout the session, after departures from idle periods. For the full details of our analysis see Appendices 3.8.1-3.8.4. For each of the service models our analysis is structured according to the following approach:

1. Model the distribution of periods as Coxian phase-type distributions, to capture the general distributions that can occur in real systems (Appendix 3.8.1).

2. Construct a Continuous-Time Markov Chain (CTMC) modeling session arrivals, the state of every session at a phase-level of detail, the transitions between phases of sessions, and the transitions leading to session termination. (Appendix 3.8.1).

3. Derive the stationary distribution of the system using local balance equations. This is a common approach in queuing theory (see e.g. [21]) for deriving the stationary distribution of a CMTC (Appendix 3.8.3).

4. Use the stationary distribution to obtain the probability that resources are occupied according to a particular state vector, when the system is in steady-state. Use these probabilities to derive the probability that an arriving request is blocked. Simplify

Symbol	Definition	Typical Range
System Parameters		
K	Number of nodes connected to the EGS	$\mathbb{Z}^+ = \{0, 1, 2, \cdots\}$
c_k	Number of communication qubits at node n_k	$\{1, 2, 3, \cdots\}$
C	Total number of EGS resources	$\{1, 2, 3, \cdots\}$
T	Number of session types	$\{1, 2, 3, \cdots\}$
F	Number of possible flows, $F = \lvert\mathcal{F}\rvert$, where $\mathcal{F}$ is the set of all possible flows	$\binom{K}{2} \times T$
Flow-level quantities		
$f,\ f_j$	Flow identifier	$f \in \mathcal{F}$
$n_k \in f$	Node n_k partakes in flow f	–
ν_1^f	Session arrival rate for flow f	$[0, \infty)$
ρ^f	Traffic intensity for flow f	$[0, \infty)$
q_j	Number of active sessions for flow f_j	$\mathbb{Z}^+$
$\mathcal{S}$	System state-space (a set of system states possible within a service model)	–
$\mathbf{q}$	System state vector: $[q_1,\ q_2, \cdots,\ q_F]$	$\mathbf{q} \in \mathcal{S}$
$\mathcal{A}_j^f$	The j^{th} attempt period for flow f	$[1, M_A^f]$
$M_A^f / M_C^f / M_I^f$	Number of attempt/calibration/idle periods	$\mathbb{Z}^+$
$1/\eta_{A,j}^f$	Mean duration of the j^{th} attempt period of a session for flow f	$[0, \infty)$
$1/\eta_{C,j}^f$	Mean duration of the j^{th} calibration period of a session for flow f	$[0, \infty)$
$1/\eta_{I,j}^f$	Mean duration of the j^{th} idle period of a session for flow f	$[0, \infty)$
Blocking probabilities		
$\overline{\pi}_f(C)$	Blocking probability for flow f when the EGS has C resources	$[0, 1]$
$\overline{\pi}(C)$	Average blocking probability when the EGS has C resources	$[0, 1]$

Table 3.2: Summary of the primary, period level notation used to present the results of our analysis. In Appendix 3.8.1 phase level notation is defined, which is used in the proofs of our results.

by aggregating over phase-level states to obtain results to a period-level of detail. (Appendix 3.8.4)

5. Prove that an insensitivity result applies (Appendices 3.8.4 and 3.8.4): the probability that a request is blocked depends only on $\rho^f = \nu_1^f \times \mathbb{E}[\text{session duration}]$, rather than on the detailed distributions of period durations.

3.5.1. Single EPR Pair Generation with Strict Resource Reservation

Recall from Section 3.4.3 that in this service model, sessions consist of batches of attempt periods interleaved with calibration periods. These sessions do not have any idle periods, so that $M_I^f = 0\ \forall f \in \mathscr{F}$. Once a session is admitted, it holds onto EGS and local resources until session termination, which either occurs upon the first successful entanglement generation attempt or after all attempts are exhausted. The following theorem states the probability that an arriving request for a session of type f is blocked.

Theorem 8. *For the system with single EPR pair generation operating in strict resource reservation mode, the probability that an arriving request belonging to flow $i \in \{1, \ldots, F\}$ is*

blocked is given by

$$\overline{\pi}_i(C) = \left(\sum_{\mathbf{q} \in \mathcal{Q}'(i)} \prod_{j=1}^{F} \frac{\left(\rho^{f_j}\right)^{q_j}}{q_j!} \right)^{-1} \sum_{\mathbf{q} \in \mathcal{Q}(C) \cap \mathcal{Q}'(i)} \prod_{j=1}^{F} \frac{\left(\rho^{f_j}\right)^{q_j}}{q_j!}, \tag{3.7}$$

where $\mathbf{q} = [q_1, \ldots, q_F]$ *represents the number of active sessions* q_i *from each flow* f_i*, the overall traffic intensities* $\rho^{f_j}\ \forall\, j$ *are given by* (3.3) *and* (3.4)*,*

$$\mathcal{Q}(h) \coloneqq \left\{ \mathbf{q} \in \mathbb{Z}_+^F : \sum_{i=1}^{F} q_i = h, \sum_{i:n_k \in f_i} q_i \leq c_k, \forall\, k \in \{1, \ldots, K\} \right\}, \text{ and} \tag{3.8}$$

$$\mathcal{Q}'(i) \coloneqq \left\{ \mathbf{q} \in \mathbb{Z}_+^F : \sum_{j:n_k \in f_j} q_j \leq c_k, \forall\, n_k \notin f_i, \sum_{j:n_l \in f_j} q_j < c_l, n_l \in f_i \right\}, \tag{3.9}$$

where $\mathbb{Z}_+$ *indicates the set of non-negative integers.*

The set $\mathcal{Q}(h)$ contains all possible combinations of active sessions such that the total number of active sessions is exactly h, and communication qubit constraints are not violated. The set $\mathcal{Q}'(i)$ contains all combinations of active sessions such that communication qubit constraints are not violated, with the additional constraint that nodes belonging to flow f_i have at least one unoccupied communication qubit each.

Proof. See Appendix 3.8.4 for the full proof. Here we sketch the idea of the proof.

Let $\mathrm{P}(\mathbf{q})$ denote the probability that EGS resources are occupied according to $\mathbf{q}$. Consider the following two events: $\Omega_1(h)$ is the event that h EGS resources are occupied, and $\Omega_2(i)$ is the event that flow f_i has available communication qubits. By the PASTA (Poisson Arrivals See Time Averages) property, we can write the probability that an arriving request of flow f_i sees h occupied resources (conditioned on the flow having enough qubits to generate a request), i.e., $\mathrm{P}(\Omega_1(h)|\Omega_2(i))$, as

$$\begin{aligned} \overline{\pi}_i(h) &= \frac{\mathrm{P}(\Omega_1(h) \cap \Omega_2(i))}{\mathrm{P}(\Omega_2(i))} \\ &= \left(\sum_{\mathbf{q} \in \mathcal{Q}'(i)} \mathrm{P}(\mathbf{q}) \right)^{-1} \sum_{\mathbf{q} \in \mathcal{Q}(h) \cap \mathcal{Q}'(i)} \mathrm{P}(\mathbf{q}). \end{aligned} \tag{3.10}$$

Here we use the fact that flow f_i would not be able to initiate a session if any node associated with f_i does not have an unoccupied communication qubit.

For a given flow-level state vector $\mathbf{q}$, there may be several phase-level state vectors $\mathbf{x}$ (introduced in Appendix 3.8.1) compatible with describing the system configuration. The probability $\mathrm{P}(\mathbf{q})$ that flows occupy EGS resources according to $\mathbf{q}$ can be calculated by summing over the state dependent stationary distributions of the system $\pi(\mathbf{x})$ (derived in Appendix 3.8.3) compatible with the flow-level state $\mathbf{q}$. The final result eventually follows by substitution into (3.10) with $h = C$ and simplification of the resulting expression.

□

Using Theorem 8 we can derive an expression for the average blocking probability for an incoming request by taking an expectation over flow-type of the incoming request. For the following, let $\overline{\pi}_{f_i}(C) \equiv \overline{\pi}_i(C)$, where flow $f_i \in \mathscr{F}$ corresponds to the ith flow label in $\{1,\ldots,F\}$.

Corollary 1. *The average blocking probability of an incoming entanglement request, denoted by $\overline{\pi}(C)$, is given as*

$$\overline{\pi}(C) = \sum_{f\in\mathscr{F}} \left\{ \frac{\mathrm{P}(\mathscr{Q}'(f))\nu_1^f}{\sum_{g\in\mathscr{F}} \mathrm{P}(\mathscr{Q}'(g))\nu_1^g} \right\} \overline{\pi}_f(C). \tag{3.11}$$

Proof. Recall that $\mathscr{Q}'(f)$, given by (3.9), is the set of all combinations of active sessions such that communication qubit constraints are not violated, with the additional constraint that nodes belonging to flow f have at least one unoccupied communication qubit each. For a flow f, entanglement requests are generated according to a Poisson process with rate ν_1^f only when all the associated nodes have communication qubits, which happens with probability $\mathrm{P}(\mathscr{Q}'(f))$. If an entanglement request is triggered then the probability that it is of type f is proportional to $\mathrm{P}(\mathscr{Q}'(f))\nu_1^f$. From Theorem 8 we have the expression for the blocking probability for a type f request as $\overline{\pi}_f(C)$. Therefore the average blocking probability is given by (3.11). □

Finally, it remains to prove the insensitivity of each flow's blocking probability to the traffic characteristics of the system beyond the flow-level traffic intensities.

Theorem 9. *The blocking probabilities*

$$\overline{\pi}_i(C),\ i \in \{1,\ldots,F\}$$

for the system with single EPR pair generation and strict resource reservation depend only on the mean traffic intensities at the flow level, ρ^{f_i}, and are not sensitive to the underlying distributions of attempt and calibration period duration.

To prove the insensitivity theorem we show that the stationary distribution of the Markov chain that describes the system behaviour remains the same when attempt and calibration durations have Coxian or exponential distributions, as long as the means of the distributions match. This implies that the result also holds for a general distribution. A detailed proof of this theorem is presented in Appendix 3.8.4.

3.5.2. Multiple EPR Pair Generation with Strict Resource Reservation

When the generation of multiple EPR pairs is permitted in strict resource reservation mode, each session, if admitted for service, traverses all periods. Consequently, session termination only occurs from the final attempt period of the session. The overall system is thus very similar to that of Section 3.4.3, and all previous assumptions hold, with the exception of early termination following a successful entanglement generation attempt.

The result is that in this service model the probability of reaching the j^{th} period is always 1. This modifies the "overall traffic intensity" from that given by (3.3) and (3.4) to

$$\begin{aligned}\rho^f &= \nu_1^f \times \mathbb{E}[\text{session duration}] \\ &= \nu_1^f \times \Big(\sum_{j=1}^{M_A^f} \frac{1}{\mu_{A,j}^f} + \sum_{j=1}^{M_C^f} \frac{1}{\mu_{C,j}^f} \Big). \end{aligned} \tag{3.12}$$

Following these considerations, in the multiple EPR pair generation with strict resource reservation service model the blocking probability for a flow f_i, $i \in [1, \cdots, F]$ is given by (3.7), where $\forall f \in \mathscr{F}$, ρ^f is given by (3.12).

3.5.3. MULTIPLE EPR PAIR GENERATION WITH RESOURCE RELINQUISHMENT

Recall from Section 3.4.3 that in this service model, sessions consist of batches of attempt periods interleaved with idle periods, at the start of which EGS resources are relinquished. Following an idle period, a session must re-attempt to secure an EGS resource. If no resource is available during such a retrial, then the request is blocked and the session jumps-over the intended batch of attempts and transitions to the next idle period, if there are any remaining. If no idle periods remain, then such a failed retrial terminates the session. If instead the retrial following the last idle period succeeds, then termination of the session is triggered by completion of the final attempt in this final batch of attempts. These sessions do not have any EGS resource retaining calibration periods, so that $M_C^f = 0 \; \forall f \in \mathscr{F}$.

The blocking probability for this service mode is of the same form as (3.7) for the two previously discussed service models, with the main differences again manifesting through the traffic intensities ρ^{f_i}. Additional consideration is needed within the analysis to account for the fact that blocking may occur not only at the beginning of a session, but also during – following idle periods. We will show that for any given session, the probability of being blocked at the beginning of the session equals the probability of being blocked immediately after an idle period.

Theorem 10. *For the system with jump-over blocking service mode, the probability that a request belonging to flow $i \in \{1, \ldots, F\}$ is blocked at the beginning of a session or during retrials of a session is given by*

$$\overline{\pi}_i(C) = \left(\sum_{\mathbf{q} \in \mathscr{Q}'(i)} \prod_{j=1}^{F} \frac{\left(\rho^{f_j}\right)^{q_j}}{q_j!} \right)^{-1} \sum_{\mathbf{q} \in \mathscr{Q}(C) \bigcap \mathscr{Q}'(i)} \prod_{j=1}^{F} \frac{\left(\rho^{f_j}\right)^{q_j}}{q_j!}, \tag{3.13}$$

where $\mathbf{q} = [q_1, \ldots, q_F]$ represents the number of active sessions q_i from each flow f_i,

$$\mathscr{Q}(h) \coloneqq \Big\{ \mathbf{q} \in \mathbb{Z}_+^F : \sum_{i=1}^{F} q_i = h, \sum_{i : n_k \in f_i} q_i \leq c_k, \forall k \in \{1, \ldots, K\} \Big\}, \text{ and}$$

$$\mathscr{Q}'(i) \coloneqq \Big\{ \mathbf{q} \in \mathbb{Z}_+^F : \sum_{j : n_k \in f_j} q_j \leq c_k, \forall n_k \notin f_i, \sum_{j : n_l \in f_j} q_j < c_l, n_l \in f_i \Big\},$$

where $\mathbb{Z}_+$ indicates the set of non-negative integers.

Proof. We provide a proof in Appendix 3.8.4. The proof relies on the phase-level system model introduced in Appendix 3.8.1, and the derivation of the stationary distribution of the system in this service model given in Appendix 3.8.3. □

Intuitively, (3.13) denotes the conditional blocking probability that all EGS resources are busy when the users of the flow f_i make an attempt to initiate a session or trigger retrials during a session. Eq. (3.13) represents the ratio of the probability that both nodes of flow f_i have unoccupied communication qubits and that all EGS resources are occupied (numerator), to the probability that nodes of flow f_i have unoccupied communication qubits to be able to participate in a session (denominator). We remark that *a priori* there is no reason to believe the the blocking probability for the first attempt of a session and subsequent retrials within that session will obey the same formula. Our result, that (3.13) applies equally to each of these cases, ensures that in practice it is straightforward to evaluate the blocking probability.

3.6. Numerical Evaluation

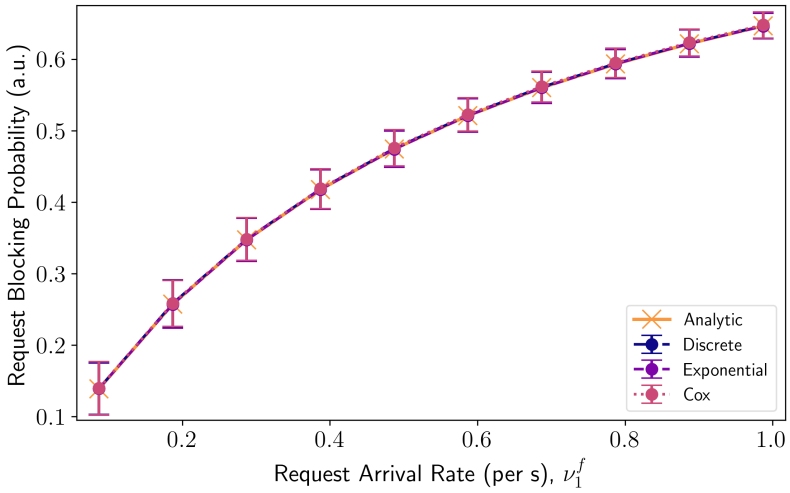


Figure 3.7: Comparison of the average blocking probability per flow according to (3.7) with simulations for an EGS with one resource, connected to eight nodes, each with a single communication qubit. The EGS serves $\binom{8}{2} = 28$ flows, one for each possible node pairing. Session traffic is homogeneous. The absolute relative errors are $(\delta_{\text{discrete}}, \delta_{\text{exponential}}, \delta_{\text{Cox}}) = (0.004, 0.001, 0.003)$.

In real-world implementations of entanglement generation, every individual attempt and calibration period has a finite duration. In a demonstration of deterministic HE delivery carried out between two NV nodes [42], for instance, the authors describe entanglement generation attempts as taking a fixed amount of time, $\Delta t_{\text{attempt}}$. On the other hand, the calibration periods take a variable amount of time, of which the mean duration μ_{calib} is known. To model such an experiment one could sample the duration of each

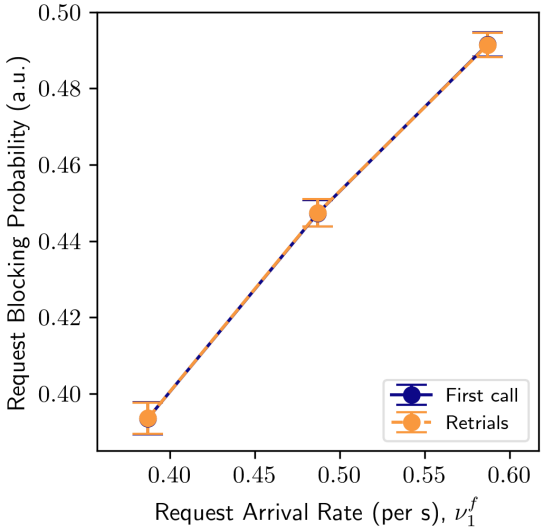


Figure 3.8: Average blocking probability for the first call of a session in the jump-over service model compared to that of calls which follow idle periods (retrials). Data is obtained from discrete simulations of an EGS with one resource, serving eight nodes via $\binom{8}{2} = 28$ flows. Every node is restricted to a single communication qubit. Session traffic is homogeneous. The maximum absolute relative difference between the average blocking probabilities is 0.00025.

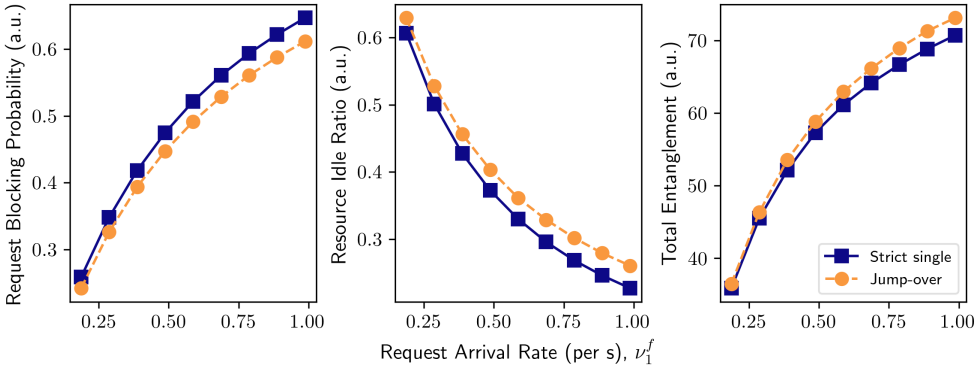


Figure 3.9: Comparison of the strict single and jump-over service models, for an EGS with one resource, serving eight nodes via $\binom{8}{2} = 28$ flows. Every node is restricted to a single communication qubit. Within each service model, session traffic is homogeneous. Left: blocking probability per request. Middle: proportion of time that the EGS resource is idle, compared to total simulation time. Right: total amount of entanglement generated by all sessions during the time simulated. Data is obtained using discrete simulations.

calibration period from an exponential distribution with mean μ_{calib} and fix the duration of attempts to $\Delta t_{\text{attempt}}$.

To validate our analysis, we simulate three models of an entanglement generation experiment. These are referred to as *discrete*, *Cox*, and *exponential* and are differentiated by how the duration of entanglement generation attempts and calibration periods are

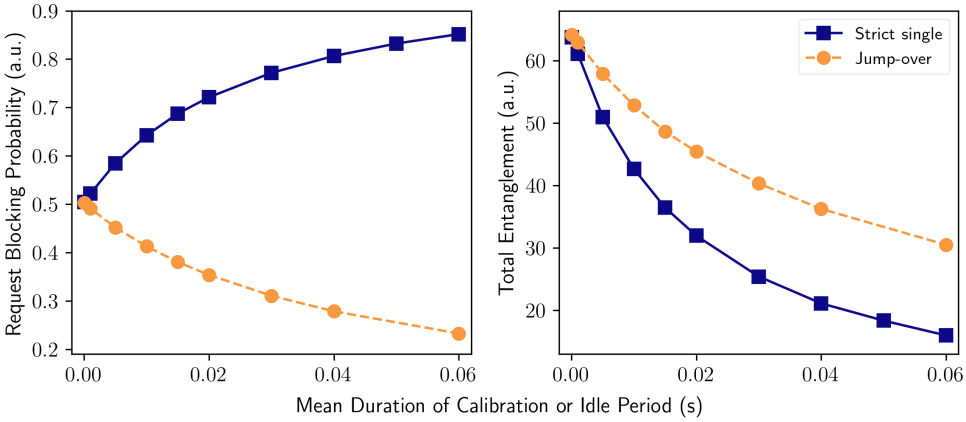


Figure 3.10: Comparison of the effect of varying the mean duration of calibration periods (strict single service model) and idle periods (jump-over service model). Within each service model, session traffic is homogeneous with a fixed request arrival rate of 0.587 per second. The EGS has one resource and serves eight nodes via $\binom{8}{2} = 28$ flows. Every node is restricted to a single communication qubit. Data is obtained using discrete simulations.

determined. To ensure the simulations are compatible with our analysis two key assumptions are made. First, in all simulation modes and in numeric evaluation of (3.7) we fix the mean duration of every attempt (calibration period/idle period) to some value T_{attempt} ($T_{\text{calib}}/T_{\text{idle}}$). In discrete simulations the duration of each attempt (calibration period/idle period) is set exactly to these values. Settings (number of phases, duration of phases, transition probabilities between phases) for the Cox distribution, are chosen to ensure (3.55) is satisfied. Our insensitivity result suggests these settings may be chosen arbitrarily, as long as they obey (3.55). In Appendix 3.8.6 we include a sample of parameters used for the Cox distribution governing attempts. The second assumption is that the probability any attempt results in successful entanglement generation is a fixed value, p_{gen}. For justification, see Section 3.3.1. Simulation parameters are detailed in Table 3.3. Further details on simulation implementation are included in Appendix 3.8.6.

To quantify agreement between numeric evaluation of (3.7) and simulated results we define error parameters $\delta_{\text{sim. type}}$ based on the maximum absolute relative difference between the points of the analytic and simulated data sets,

$$\delta_{\text{sim. type}} := \frac{\left|\max_x(y_{\text{Analytic}}[x] - y_{\text{Sim.}}[x])\right|}{y_{\text{Analytic}}[\arg\max_x(y_{\text{Analytic}}[x] - y_{\text{Sim.}}[x])]}, \tag{3.14}$$

where y_{Analytic} denotes an analytic data set, $y_{\text{Sim.}}$ denotes a simulated data set, and square brackets denote indexing the data sets. The error parameter reports the difference between the analytic and simulated data point for which the difference is maximum, relative to the analytic value at that point.

Each simulation is run for a duration corresponding to 1e7 iterations of the discrete simulation, equivalent to a simulation of 1150.73 seconds of *simulated real-time*. See Appendix 3.8.6 for details of the expected number of requests placed over the duration of a simulation. For any run of a simulation, the average blocking probability is calculated

Description	Value
Link lengths	10 km
One-way communication time (RTT/2)	50.03 μs
Duration of calibration period (CR check), T_{calib}	1 ms
Probability single attempt success, p_{gen}	1e-5 (a.u.)
Duration single attempt, T_{attempt}	115.072 μs
Attempt batch size	100
# batches (strict allocation) or re-trials (jump-over)	10
# calibration (idle) periods in strict (jump-over) mode	9

Table 3.3: Physical parameters used in simulations correspond to an EGS supporting batched single click HE generation for NV colour centers in diamond as quantum nodes. We use a pessimistic value of p_{gen} as compared with published values (e.g., [42]) for the purpose of absorbing the additional sources of loss, e.g., that which may be introduced by insertion of an optical switch in the optical paths from nodes to EGS resources.

by averaging the blocking probabilities recorded by each flow. Every data point from a simulation is the result of averaging over 200 independent runs of the simulation. Error bars for request blocking probabilities correspond to the average (over the runs) standard deviation in blocking probabilities between flows.

Traffic in a deployed network will in general be non-homogeneous. The causes of non-homogeneity may stem from differences in the physical parameters of quantum nodes and their connections to an EGS or they may result from differences in request parameters submitted by flows, enabling them to target different applications. In general, our analysis applies to non-homogeneous traffic. However, the simplest physical implementation of the EGS is one where all nodes are connected to the hub by links of equal length and all flows submit identical requests for sessions with equal rates. This is a homogeneous traffic scenario. In the remainder of this section we first present results from each of the three service models developed in Section 3.4 in a homogeneous traffic scenario. Then, we demonstrate the application of our analysis to a non-homogeneous traffic scenario in the single entanglement generation with strict resource reservation service model.

3.6.1. HOMOGENOUS TRAFFIC

In a deployed quantum network, a network operator may be interested in selecting a service model for an EGS based on its performance across a range of metrics. We compare the performance of the three service models of Section 3.4. In what follows, these service models are referred to simply as *strict single*, *strict multiple* and *jump-over*, respectively. Besides request blocking probabilities, we also study resource utilization and the total entanglement generated in a fixed amount of time. The former provides information on how efficiently network resources are used, and the latter gives a measure of the productivity derived from the allocation of network resources. To study the resource utilization in simulation, we define the *resource idle ratio* as the proportion of time that one or more EGS resources is idle, relative to the entire simulated time. To study the total entanglement generated we track and sum the successful generation of entangled pairs by any session over the duration of the entire simulated time.

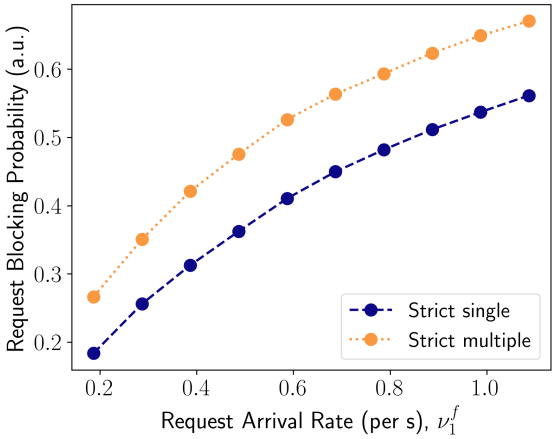


Figure 3.11: Comparison of the request blocking probabilities of the strict single and strict multiple service modes when there is a high probability ($p_{\text{gen}} = 0.001$) that an attempt to generate entanglement succeeds. Data is obtained from discrete simulations of an EGS with one resource, serving eight nodes with $\binom{8}{2} = 28$ flows. Every node is restricted to a single communication qubit. Session traffic is homogeneous. The absolute relative errors are $(\delta_{\text{discrete}}, \delta_{\text{exponential}}, \delta_{\text{Cox}}) = (0.014, 0.018, 0.003)$ for the strict single service mode and $(0.032, 0.015, 0.002)$ for the strict multiple service mode.

Before comparing the service models, we validate our analysis of the blocking probability for each of the three service models. Figure 3.7 compares numeric evaluation of (3.7) with results from discrete, exponential, and Cox simulations of an EGS operated in the strict single service model, with control of one resource ($C = 1$), connected to eight nodes, each with a single communication qubit. Load on the EGS network results from requests for resource access by the flows. In alignment with expectations, we observe that the request blocking probability increases as the load on the network increases. Close agreement is observed between the analytic and simulated results, all of which overlap well within one standard deviation for every data point. This is expected due to the insensitivity result (Theorem 9) and supports our analysis. The tightness of the overlap between each simulated data set and the analytic results is captured by the absolute relative errors, defined by (3.14). These errors are $< 1\%$ for each simulation type. We validate the other service models in the same way and obtain error parameters of $(\delta_{\text{discrete}}, \delta_{\text{exponential}}, \delta_{\text{Cox}}) = (0.061, 0.015, 0.006)$ for the multiple EPR pair service model and $(0.001, 0.003, 0.004)$ for the jump-over service model. To further validate our analysis of the jump-over service model, Figure 3.8 compares the average probability that the first call of a session is blocked with the probability that any call which follows an idle period (retrial) is blocked. The comparison is made using results from discrete simulations, however similar results are obtained from exponential and Cox simulations. As predicted by our analysis (Theorem 10), the blocking probability is equal for any arbitrary call and does not depend on whether a call is the first one of a session or follows an idle period (retrials). This property simplifies application of our analytic results, as only one formula needs to be evaluated.

The strict reservation service models (strict single, strict multiple), where the batches of entanglement generation attempts in a session are separated by calibration periods

during which nodes retain EGS resources, and the jump-over service model, where these batches of attempts are separated by idle periods during which nodes release EGS resources, are first contrasted in Figure 3.9. There, the performance of the strict single and jump-over service models is reported for various request arrival rates. For the physically motivated simulation parameters used (Table 3.3) in these evaluations the results for the strict single and strict multiple service models exactly coincide, hence the strict multiple data points are omitted from Figure 3.9 (and similarly for Figure 3.10). To facilitate meaningful comparison, the mean duration of the calibration periods of the strict reservation service models and the idle periods of the jump-over service model are all set equal to 1 ms (the same value as in Table 3.3). Then, to emphasize the impact of resource relinquishment, the performance of the service models is contrasted for various durations of the calibration or idle periods in Figure 3.10. In that comparison the load on the network was set equally for each service model, by setting the rate of request arrivals from each flow to a value of 0.587 per second. The data in Figures 3.9 and 3.10 is obtained from discrete simulations, but exponential and Cox simulations were also conducted. The absolute relative errors as defined by (3.14) were $\ll 1\%$ for all simulation types of each service model.

For the EGS configuration investigated, the jump-over service mode makes more efficient and productive use of the EGS resources than the strict reservation service modes. With respect to the performance under varied load (Figure 3.9), when there is a relatively low-load on the EGS, the difference between each performance metric of the jump over and strict reservation service models is marginal. When there is high load on the EGS, the jump-over service model results in lower blocking probabilities, indicating better handling of the high load. The lower blocking probability of the jump-over service model is reflected in the increased idle time ratio. Interestingly, although the EGS resources are idle for a greater proportion of the time in the jump-over mode, a greater total amount of entanglement is produced. The positive impact of resource relinquishment on the total amount of entanglement produced is further exposed when the calibration and idle period durations are varied (Figure 3.10). Regardless of the service model, the greatest amount of entanglement produced in a fixed time occurs when the mean duration of calibration or idle periods are zero. As the mean duration of calibration periods in the strict reservation service modes increases, the amount of entanglement produced in a fixed time decreases dramatically and the blocking probability approaches 1 asymptotically. In the strict reservation models, these effects follow from sessions retaining use of the EGS resources for the long durations of calibration periods, preventing the generation of entanglement by other sessions during these periods and resulting in blocking of any requests that arrive during these periods. In contrast, as the mean idle or calibration period duration increases, in the jump-over service mode there is a significantly lower rate of decrease in the total amount of entanglement produced. In complement, the request blocking probability decreases towards 0 asymptotically as the mean duration of an idle period increases. In the jump-over service model, these effects arise because requests for sessions arriving during long idle periods of an active session may be accepted. An increase in the number of active sessions contributes positively to the rate of entanglement generation, partially compensating for the lack of entanglement produced during long idle periods.

For the physically motivated simulation parameters used (Table 3.3), the performance

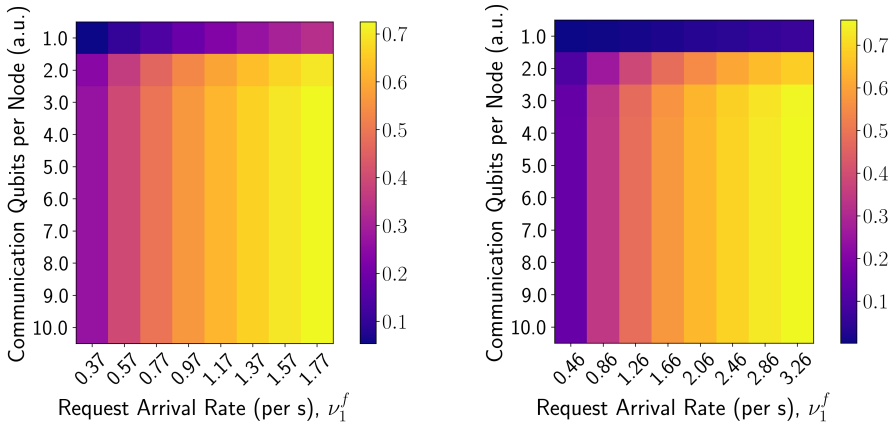
of the strict multiple service model is not significantly different from that of the strict single service model. With these parameters, the expectation value of successful entanglements per session is 0.01 EPR pairs.

The differences between the strict single and strict multiple service models are more evident if the probability that an attempt to generate entanglement succeeds is significantly increased. In Figure 3.11 we compare the blocking probabilities resulting from discrete simulation of these service modes when $p_{\text{gen}} = 0.001$, an increase of two orders of magnitude relative to our baseline simulation parameters. The expectation value of successful entanglements per session is 1 EPR pair. In this setting, the strict single service model realizes a much lower blocking probability than the strict multiple service model. This difference can be understood as resulting from a difference in the mean service times of the sessions. The recorded mean service time of sessions in the strict multiple service model is 125 ms. In the strict single service model however, sessions may end earlier if an attempt at entanglement generation succeeds, resulting in a lower observed mean service time of 78 ms. The shorter mean service time of sessions in the strict single service model has the effect that, for the same session request arrival rate, resources are free more often, leading to a lower blocking probability.

To investigate the impact of the restrictions on communication qubits, we numerically evaluated the blocking probability for an EGS controlled by the strict single service model with two or three resources (Figure 3.12) as the restriction varies from one to ten communication qubits per node. In each case, for any fixed request rate we observe that an increase in the number of communication qubits per node up to the number of EGS resources results in a large increase in the blocking probability, but further increases in the number of communication qubits have little impact. Numeric evaluation for strict service model scenarios where an EGS with one, two, or three resources is connected to twenty nodes and serves $\binom{20}{2} = 190$ flows (see Appendix 3.8.7) confirm that the same effect holds for an EGS serving a large number of flows. Recall that a flow does not submit a request if one or more of the nodes of the flow does not have a communication qubit available. Hence for any fixed request rate, a lower blocking probability is expected if nodes do not have enough communication qubits available, due to the lowered effective request rate. As a result of these properties and the observed trends, we conclude that for the EGS configurations studied, when each node is restricted to a number of communication qubits less than or equal to the number of EGS resources (one or two in these configurations), resource contention at the EGS has less of an impact on the blocking probability than does a lack of available communication qubits. In contrast, when nodes are equipped with a number of communication qubits in excess of the number of EGS resources (three or more in these configurations), contention for EGS resources becomes the limiting factor affecting the blocking probability.

3.6.2. NON-HOMOGENEOUS TRAFFIC

Non-homogeneity resulting from network topology will be relevant in any deployed network, as it is unrealistic to expect that nodes may be located in a perfect disk with fixed link lengths to an EGS. We examine a situation where non-homogeneous traffic in the strict single service model results from a network topology where half of the nodes are connected to the EGS by links of length 10 km and the other half of the nodes, are



(a) two EGS Resources ($C = 2$)

(b) three EGS Resources ($C = 3$)

Figure 3.12: Heatmaps of the average blocking probability per flow when the number of communication qubits per node and the request arrival rates are varied. Data results from numeric evaluation of (3.7) for an EGS with eight nodes and serving $\binom{8}{2} = 28$ flows, one for each possible node pairing. Session traffic is homogeneous.

Description	Value
Link lengths: users $1 - \lceil K/2 \rceil$	10 km
Link lengths: users $(\lceil K/2 \rceil + 1) - K$	20 km
Single-mode optical fiber attenuation coefficient, σ_{att}	0.2 dB/km
One-way communication time, S_2 (RTT/2)	100.07 μs
Resource allocation duration, S_1	125 ms
Resource allocation duration, S_2	240 ms
Probability photon travelling 1 km optical fiber arrives, $p_a = 10^{-(\sigma_{\text{att}}/10)}$	0.95499 (a.u.)
Probability of single attempt success in S_2: $p_{\text{gen}} \cdot p_a^{10}$	6.31e-6 (a.u.)
Duration of a single attempt, S_2	230.146 μs

Table 3.4: Additional physical parameter settings for a non-homogeneous traffic scenario where half of the nodes have doubled link lengths to the EGS. These parameters supplement those of Table 3.3.

connected by links of double the length, i.e. 20 km. Flows are partitioned into two sets, labelled as *set 1* (S_1) and *set 2* (S_2). For every flow in S_2, at least one of the nodes is connected to the EGS by a 20 km link. For every flow in S_1, both nodes are connected to the EGS by 10 km links. The length of a link between a node and the EGS affects the RTT and the probability that a photon sent over the link arrives at the EGS. Table 3.4 recounts the physical parameters for each set of nodes. See Appendix 3.8.6 for further discussion of the simulation implementation details. Note that within each set of flows, traffic is homogeneous.

Figure 3.13 demonstrates that an arbitrary flow receives different service depending

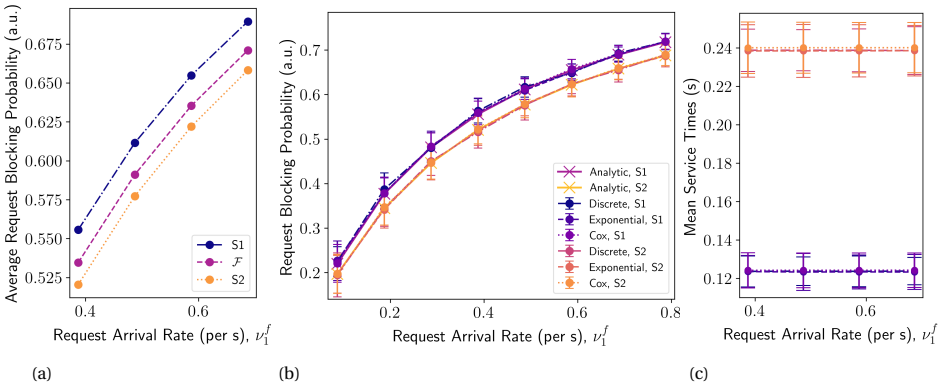


Figure 3.13: Non-homogeneous traffic, strict single service model: Four nodes are connected to an EGS by 10 km links and four by 20 km links. The EGS has 1 resource. The set $\mathcal{F}$ is partitioned into sets S_1 and S_2. For every flow in S_2, at least one of the nodes is connected to the EGS by a 20 km link. For every flow in S_1, both nodes are connected to the EGS by 10 km links. Traffic within each set is homogeneous. (a) Analytic blocking probability for flows within sets S_1 and S_2 compared to that of an arbitrary flow $f \in \mathcal{F}$, as given by (3.11). (b) Average request blocking probabilities within sets S_1 and S_2 from discrete, exponential, and Cox simulations compared with numeric evaluation of (3.7). The absolute relative errors are $(\delta^{S_1}_{\text{discrete}}, \delta^{S_1}_{\text{exponential}}, \delta^{S_1}_{\text{cox}}) = (0.027, 0.032, 0.0032)$ and $(\delta^{S_2}_{\text{discrete}}, \delta^{S_2}_{\text{exponential}}, \delta^{S_2}_{\text{cox}}) = (0.054, 0.0088, 0.0064)$. (c) Comparison of mean service times for sessions in S_1 with those in S_2. Legend is the same as in (b).

on whether it is an element of set S_1 or S_2. Requests from flows in S_1 are blocked with a higher probability and have shorter mean service times than flows in S_2. The blocking probability of any arbitrary flow f from the set of total flows $\mathcal{F}$, indicated in Figure 3.13a, is the average request blocking probability as given by (3.11). For the scenario investigated, the blocking probability of an arbitrary flow is greater than for flows in S_2 but less than for flows in S_1. The gap between the blocking probability for flows in S_1 and S_2 is due to the restriction to a single communication qubit per node and the difference in service times of sessions for flows in each set. Although flows from S_1 and S_2 have equal request rates, due to the longer mean service times of flows in S_2, one or more nodes from flows in S_2 will not have a free communication qubit more often than for flows in S_1. As a result, the effective request rate and hence the blocking probability is lower for flows in S_2. The absolute relative errors between the average blocking probability of each simulation type and the analytic blocking probability are $< 4\%$ ($< 6\%$) for flows in S_1 (S_2). These results validate (3.7) in the presence of non-homogeneity and highlight the complex interplay between the number of communication qubits at a node, mean service times of a session, and blocking probability of an arriving session.

3.7. Conclusion and Future Work

We have proposed an on-demand resource allocation algorithm for an EGS and developed its performance analysis in a variety of traffic scenarios and operation modes. We modeled the system as an Erlang loss one, and discovered an insensitivity property for the request blocking probability. Numerical results validate our analysis. Our work can lead to

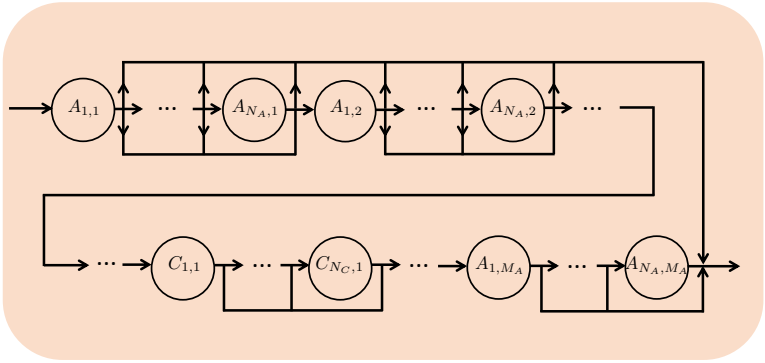


Figure 3.14: A session from the single EPR pair generation with strict resource reservation service model, shown at the level of periods in Figure 3.5, decomposed into exponentially-distributed phases so as to result in Coxian-distributed attempt and calibration periods.

several new research directions; we provide a non-exhaustive list here. The analytic and simulation frameworks we provide are valuable tools for the development of load-balancing control algorithms for an EGS, which could run at a higher level in the control stack to ensure stable quality of service can be delivered to flows. An important highlight of our model is that it flexibly incorporates restrictions that are very present in NISQ era quantum devices, hence being relevant for the development of a real near-term network. This feature of the model can be used as a tool to investigate efficient resource provisioning schemes – not only for a single EGS serving a number of nodes in a star topology, but also for a more complex network made up of heterogeneous devices. In future work, one could allow a flow to have control over multiple communication qubits. This generalization, along with some of the modeling challenges that might be encountered, were discussed briefly in Section 3.4. Another natural extension of our model is to generalize from bipartite to multipartite entangled states. This can be accomplished by expanding the definition of a "flow" to include $k \geq 2$ nodes. Last, one could study various back-off mechanisms in lieu of blocking, and determine suitable hardware regimes for each of the schemes.

3.8. APPENDICES

3.8.1. PHASE-LEVEL SYSTEM MODEL

MOTIVATION FOR PHASE-LEVEL MODELING

In the physical systems which our model describes, attempt, calibration, and idle periods may have generally distributed durations. As discussed in Section 3.2, phase-type Coxian distributions can approximate generally distributed periods by a sequence of exponential phases that preserves the mean period duration. For examples of the distributions determining the duration of attempt and calibration periods, see the discussion based on experiments with the NV center in diamond at the beginning of Section 3.6.

To derive the blocking probability for each service model (Theorems 8 and 10) and

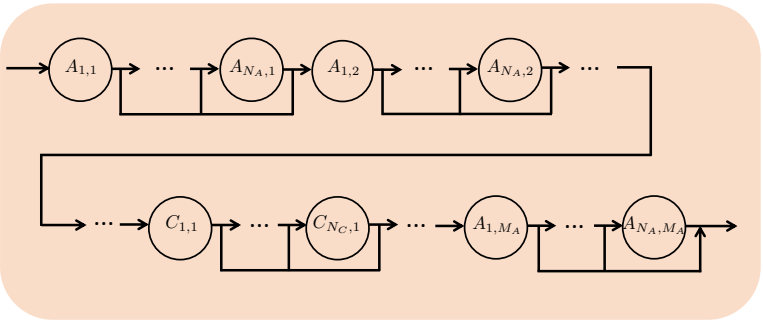


Figure 3.15: A session from the multiple EPR pair generation with strict resource reservation service model, shown at the level of periods in Figure 3.5, decomposed into exponentially-distributed phases so as to result in Cox-distributed attempt and calibration periods.

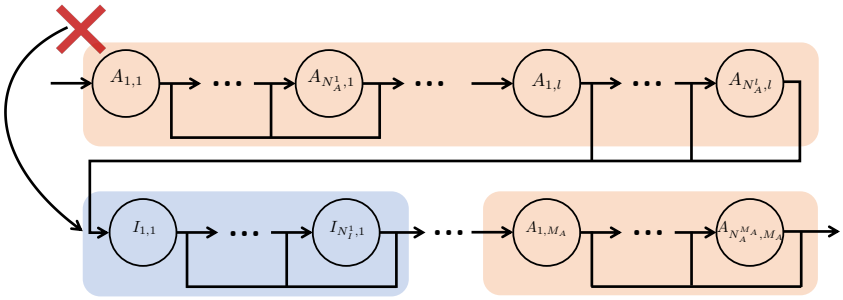


Figure 3.16: A session with jump-over blocking, shown at the level of periods in Figure 3.6, decomposed into exponentially-distributed phases so as to result in Cox-distributed active and idle periods. Transitions to outside the queueing network are permitted only from the last period of the session.

prove our insensitivity result (Theorem 9), we construct a Continuous Time Markov Chain (CTMC) model of the EGS system. This appendix defines the state space of the system for each service model, defines the traffic intensities of each flow with phase-level detail, and defines notation required to prove the results discussed in Section 3.5.

Phase-Type (Coxian) Representation

A Coxian distribution with N phases approximates a general distribution by decomposing it into N exponential stages traversed sequentially, with the possibility of exiting the sequence early, based on a transition probability between phases. Each period type in our model is represented this way:

- An attempt period is decomposed into N_A^f exponential phases;
- A calibration period is decomposed into N_C^f exponential phases;
- An idle period is decomposed into N_I^f exponential phases.

We denote individual phases as:

- $A^f_{i,j}$: the i^{th} phase of attempt period j, for a session of flow f, where $i \in \{1,\cdots,N^f_A\}$ and $j \in \{1,\cdots,M^f_A\}$;
- $C^f_{i,j}$: the i^{th} phase of calibration period j, for a session of flow f, where $i \in \{1,\cdots,N^f_C\}$ and $j \in \{1,\cdots,M^f_C\}$;
- $I^f_{i,j}$: the i^{th} phase of idle period j, for a session of flow f, where $i \in \{1,\cdots,N^f_C\}$ and $j \in \{1,\cdots,M^f_C\}$.

The total number of phases in a session of flow type f is

$$L^f \equiv N^f_A \times M^f_A + N^f_C \times M^f_C + N^f_I \times M^f_I. \tag{3.15}$$

The mean duration of the i^{th} phase of a session for flow type f, where $i \in \{1,\cdots,L\}$ is denoted $1/\mu^f_i$. Sometimes it is convenient to refer instead to the "service rate" or rate of completing the i^{th} phase of a session of flow type f, which is simply μ^f_i. If we refer to the service rate of the i^{th} phase of the j^{th} attempt/calibration/idle period of a session of flow f, the range for i changes to $\{1,\cdots,N^f_{A/C/I}\}$, and the notation $\mu^f_{i,j}$ is used instead.

The possibility of exiting the sequence of phases comprising a period j early, after the i^{th} phase, $i \in \{1,\cdots,N^f_A\}$, is described by a transition probability $1-r^f_{i,j}$. If this occurs, the period ends, and the session transitions to the subsequent period dictated by the service model, or possibly terminates in the case that the session termination condition is met.

The duration $1/\eta^f_{A,j}$ of an attempt period j for a session of flow f, which appears in the main text, can be simply related to the duration of the N^f_A exponentially distributed phases which constitute the attempt period,

$$\frac{1}{\eta^f_{A,j}} = \sum_{i=1}^{N^f_A} \frac{r^f_{i,j}}{\mu^f_{i,j}}. \tag{3.16}$$

Similarly, the same relation holds for the mean duration of calibration and idle periods, with substitution of $1/\eta^f_{A,j}$ and N^f_A for $1/\eta^f_{C,j}$ and N^f_C or $1/\eta^f_{I,j}$ and N^f_I, respectively.

PHASE-LEVEL STATE VECTOR

Irrespective of the service model, the complete state space of the system can be represented using a vector

$$\mathbf{x} = [\mathbf{x}^{f_1},\ldots,\mathbf{x}^{f_F}], \tag{3.17}$$

where F is the number of possible flows, and each $\mathbf{x}^f = [x^f_1,\cdots,x^f_{L_f}]$ tracks the number of sessions in each phase for flow f. Specifically, x^f_i is the number of sessions currently in phase i for flow type f. The dimension of this vector $\mathbf{x}$ is

$$L \equiv \sum_{f\in\mathscr{F}} L^f.$$

When we need to identify the specific period and phase within it, we write $x_{i,j}^{f,A}$ for the number of sessions in phase i of attempt period j for flow f (and similarly for calibration and idle periods). This notation is primarily used with reference to transition rates between phases and periods.

We define $\mathbf{e}_i^f, i \in \{1, \cdots, L\}$ to be a vector of dimension L with all entries zero except the one corresponding to the i^{th} component of $\mathbf{x}^f$, which is equal to one. We also use the notation $\mathbf{e}_{i,j}^{f,A}$ when specifically referring to phase i of attempt period j (and similarly for calibration and idle periods).

The flow-level state vector $\mathbf{q}$ used in Sections 3.4 and 3.5 can be simply related to the phase-level state vector $\mathbf{x}$. The number of active sessions of flow f when the system is in state $\mathbf{x}$ is

$$q^f(\mathbf{x}^f) = \sum_{i=1}^{L^f} x_i^f. \tag{3.18}$$

For service models with resource relinquishment, we distinguish between the total number of sessions and the number of active sessions, which are those currently making active use of EGS resources. The number of active sessions of flow f when the system is in state $\mathbf{x}$ is

$$\tilde{q}^f(\mathbf{x}) \equiv \sum_{j=1}^{M_A^f} \sum_{i=1}^{N_A^f} x_{i,j}^{f,A}. \tag{3.19}$$

State Space Constraints

The admissible state space depends on the service model.

(i) *Strict Resource Reservation Models:*
In all service models, the EGS must heed its resource capacity constraint (C total resource modules) and each network node must heed its own communication qubit limit (c_k communication qubits at node n_k). The following considerations apply to both the Single EPR Pair Generation with Strict Resource Reservation (Section 3.4.3) and Multiple EPR Pair Generation with Strict Resource Reservation (Section 3.4.3) service models. In these service models, all active sessions (including those in calibration periods) occupy EGS resources. Every session of flow type f occupies one communication qubit from each node in flow f. The admissible state space is thus given by

$$\mathcal{S} = \Big\{\mathbf{x} \in \mathbb{N}^L : \sum_{f \in \mathcal{F}} q^f(\mathbf{x}^f) \le C, \\ \sum_{f \in \mathcal{F}: n_k \in f} q^f(\mathbf{x}^f) \le c_k, \forall k \in \{1, \ldots, K\}\Big\}, \tag{3.20}$$

where we recall that the notation $n_k \in f$ means that node n_k partakes in flow f.

(ii) *Resource Relinquishment Model*:
The following considerations apply to the Multiple EPR Pair Generation with Resource Relinquishment (Section 3.4.3) service model. As with the strict resource reservation service models, each session of flow type f occupies one communication qubit from each node in flow f. In this service model however, only sessions in

active (non-idle) phases occupy EGS resources. The admissible state space is thus given by

$$\mathscr{S}'' = \Big\{\mathbf{x} \in \mathbb{N}^L : \sum_{f\in\mathscr{F}} \tilde{q}^f(\mathbf{x}) \le C, \\ \sum_{f\in\mathscr{F}: n_k \in f} q^f(\mathbf{x}^f) \le c_k, \forall k \in \{1,\dots,K\}\Big\}, \tag{3.21}$$

where $\tilde{q}^f(\mathbf{x})$ is given by (3.19).

Both state spaces are coordinate convex, i.e., if $\mathbf{x} \in \mathscr{S}$ (or $\mathscr{S}''$), then $\mathbf{y} \in \mathscr{S}$ (or $\mathscr{S}''$) for all $\mathbf{y}$ such that $\mathbf{0} \le \mathbf{y} \le \mathbf{x}$ component-wise.

TRAFFIC INTENSITIES

Each phase i of flow type f, $i \in \{1,\cdots,L\}$ has a service rate μ_i^f, and an arrival rate of $\lambda_i^f(\mathbf{x})$, which in general may depend on the state $\mathbf{x}$ of the system. We sometimes write $\lambda_i^f \equiv \lambda_i^f(\mathbf{x})$, when the state vector $\mathbf{x}$ is clear from context. The traffic intensity for phase i of flow type f is

$$\rho_i^f \equiv \frac{\lambda_i^f}{\mu_i^f}. \tag{3.22}$$

TRANSITIONS IN THE SINGLE EPR PAIR GENERATION WITH STRICT RESOURCE RESERVATION SERVICE MODEL

In the Single EPR Pair Generation with Strict Resource Reservation service model, the following properties apply:

- All external arrival rates (*i.e.*, those originating from outside of the network) $\nu_i^f(\mathbf{x})$ are zero, except for those to the first phase of the first attempt period of a session $A_{1,1}^f$, $f \in \mathscr{F}$. We denote these rates with $\nu_1^f(\mathbf{x})$, $f \in \mathscr{F}$, $\mathbf{x} \in \mathscr{S}$, so that the transition from $\mathbf{x}$ to $\mathbf{x} + \mathbf{e}_{1,1}^{f,A}$ occurs with rate

$$\nu_1^f(\mathbf{x}) = \begin{cases} \nu_1^f, & \text{if } \mathbf{x} + \mathbf{e}_{1,1}^{f,A} \in \mathscr{S}, \\ 0, & \text{else.} \end{cases} \tag{3.23}$$

- Transition $\mathbf{x} \to \mathbf{x} - \mathbf{e}_i^f + \mathbf{e}_{i+1}^f$ occurs with probability $p_{i,i+1}^f$, for $1 \le i < L^f$.

- A special case of the above is that $p_{i,i+1}^f = 1$, $\forall f \in \mathscr{F}$, if x_i^f corresponds to the last phase of a calibration period.

- Transition $\mathbf{x} \to \mathbf{x} - \mathbf{e}_i^f + \mathbf{e}_j^f$ occurs with probability $p_{i,j}^f$ if j is such that x_j^f corresponds to the initial phase of the attempt/calibration period that follows the attempt/calibration period corresponding to x_i^f.

- Transition $\mathbf{x} \rightarrow \mathbf{x} - \mathbf{e}_i^f$ occurs with probability p_i^f if i is such that x_i^f corresponds to an attempt phase. This transition represents the event that entanglement generation succeeds after the ith phase of flow f – in this service model, this causes the session to end. We note that leaving the session from a calibration phase is not possible.

When $i = 1$, i.e., for the first phase of a session, the total arrival rate is $\lambda_1^f(\mathbf{x}) = \nu_1^f(\mathbf{x})$. For all other phases, the arrival rate is given by

$$\lambda_i^f(\mathbf{x}) = \begin{cases} \nu_1^f \tilde{p}_i^f \equiv \lambda_i^f, \text{ if } \mathbf{x} + \mathbf{e}_i^f \in \mathscr{S}, \\ 0, \text{ else.} \end{cases} \tag{3.24}$$

Above, $\tilde{p}_i^f$, $2 \leq i \leq L^f$, denotes the probability of reaching the ith phase starting from the first phase of a session belonging to flow f. Appendix 3.8.2 provides a derivation of these probabilities. In the main text the quantities $\mathscr{P}_{A/C/I,j}^f$ refer to the probability of reaching the j^{th} attempt/calibration/idle period. This probability is equivalent to the probability of reaching the first phase of the j^{th} attempt/calibration/idle period.

A session from the Single EPR Pair Generation with Strict Resource Reservation service model is illustrated with phase level detail in Figure 3.14. In this service model, a session may terminate (transition out of the system) following any phase of an attempt period. In the figure, these transitions are illustrated by upward arrows linked to system exit.

Transitions in the Multiple EPR Pair Generation with Strict Resource Reservation Service Model

In the Multiple EPR Pair Generation with Strict Resource Reservation service model, each session, if admitted for service, traverses all periods (albeit, not necessarily all phases). Consequently, as shown in Figure 3.15, transitions to outside of the set of active sessions are only permitted from the final attempt period of the session. The overall system is thus very similar to the service model of Appendix 3.8.1, and all previous assumptions hold, with the exception that the probability p_i^f of exiting the set of active sessions in the network following a phase i is $p_i^f = 0$, $\forall f \in \mathscr{F}$, whenever i belongs to a phase other than that of the last attempt period. Recall from Section 3.4 that by convention a session always ends with an attempt period.

Transitions in the Multiple EPR Pair Generation with Resource Relinquishment Service Model

In the Multiple EPR Pair Generation with Resource Relinquishment service model, a session that had initially been admitted for service by the EGS may get blocked later on, depending on the state of the system $\mathbf{x}$ at the moment the session leaves an idle period. Thus, to define the traffic characteristics within a session, we require transition probabilities that are functions of the state. Let $p_{i,j}^f(\mathbf{x})$ be the probability of transitioning from the ith phase of a type f session to its j^{th} phase. Since in this service mode, a session can only end during its last period, the corresponding model is most similar to the "multiple EPR pair generation with strict resource reservation" scenario. Thus, all traffic characteristics of Appendix 3.8.1 apply (i.e., $p_{i,j}^f(\mathbf{x}) = p_{i,j}^f$, $\forall f, i, j, \mathbf{x}$), with the following exceptions:

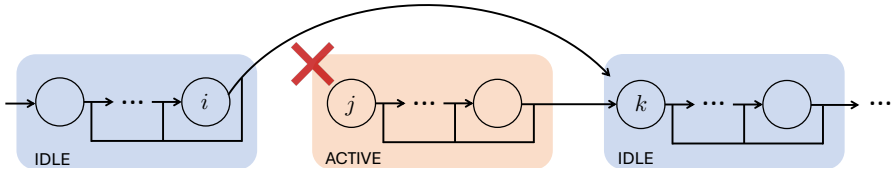


Figure 3.17: Illustration of jump-over blocking: if a transition to an active period is not possible due to a state space restriction, the active period is skipped, and a transition occurs to the first phase of the next idle period.

- If j is the starting phase of an active period (excluding the first period of a session), i is any phase of the preceding idle period, and $\mathbf{x}-\mathbf{e}_i+\mathbf{e}_j \notin \mathscr{S}''$, then $p^f_{i,j}(\mathbf{x})=0$;

- If moreover k is the starting phase of the next idle period, then $p^f_{i,k}(\mathbf{x})=p^f_{i,j}$.

These amendments describe the jump-over blocking dynamics and are illustrated in Figure 3.17.

The external arrival rates into the system are zero for all phases, except for the first of every session; these are given by $\nu^f_1(\mathbf{x})=\nu^f_1$ if $\mathbf{x}+\mathbf{e}^f_1 \in \mathscr{S}''$, and zero otherwise. Letting γ^f_i be the probability that a session belonging to flow f reaches its i^{th} phase starting from its first phase, we have that the total arrival rate into phase i for flow f while in state $\mathbf{x}$ is given by

$$\lambda^f_i(\mathbf{x})=\begin{cases}\nu^f_1\gamma^f_i \equiv \lambda^f_i, & \text{if } \mathbf{x}+\mathbf{e}^f_i \in \mathscr{S}'',\\ 0, & \text{else.}\end{cases} \tag{3.25}$$

Let us examine why λ^f_i has no dependence on the state $\mathbf{x}\in\mathscr{S}''$. For the following, suppose $\mathbf{x}+\mathbf{e}^f_i \in \mathscr{S}''$. First, consider i to be any phase of an active period, and j to be the first phase of the same period. Then $\gamma^f_i = p^f_{j,j+1}\dots p^f_{i-1,i}$. Next, let i be the first phase of an idle period. The arrival rate into this phase is ν^f_1, regardless of whether the transition is happening from the preceding active period, or from the previous idle period. The latter would happen if the system was at capacity (the EGS did not have enough resource modules) at the time of the transition, thereby causing the next active period to be skipped (along with any other active periods that immediately follow it). The new routing rules introduced above ensure that the arrival rate into the idle period is equal to that of the period(s) being jumped over. The arrival rate into any other phase of an idle period is then computed similar to that of an active period's non-initial phase.

3.8.2. PROBABILITY OF REACHING A PHASE IN SINGLE EPR PAIR GENERATION MODE

In this appendix, we derive an expression for $\tilde{p}^f_i$ as seen in (3.24), the probability of reaching the i^{th} phase of a type f session, starting from its first phase, while in strict resource reservation, single EPR pair generation service mode. Suppose that the i^{th} phase is located within the j^{th} period of the session. To have reached this phase, the

session must not have ended during any of the previous attempt phases (recall that in this service mode, sessions retain EGS resources during calibration periods). In the following, let $p^k_{i,j}$ denote the probability of transitioning from the i^{th} phase of period k to the j^{th} phase of the same period; we omit f below for cleanliness. Suppose that $p^{\text{cox}}_{\text{gen}}$ denotes the probability with which a BSM is successful under Coxian distributions. Then the probability of leaving during an attempt period k due to success in generating entanglement, conditioned on the event that the session has entered period k, is given by

$$P_k \equiv \text{P}\left(\text{leave during attempt period } k | \text{session entered period k}\right) \tag{3.26}$$

$$= p^k_1 + p^k_{1,2}p^k_2 + \cdots + (p^k_{1,2}\cdots p^k_{N^k-1,N^k}p^k_{N^k}) \tag{3.27}$$

$$= p^{\text{cox}}_{\text{gen}}, \tag{3.28}$$

where $p^k_l = (1-p^k_{l,l+1})p^{\text{cox}}_{\text{gen}}$ is the probability of the session leaving the system after phase l of period k due to a successful BSM at the EGS, and $p^k_{l,l+1}$ is the probability of transitioning to the next phase of the period. Then, the probability of the session reaching the i^{th} phase of the j^{th} period is given by

$$\text{P(session reaches } i^{\text{th}} \text{ phase of period } j, \text{ starting from the } 1^{\text{st}} \text{ phase of the session)}$$

$$= \prod_{l=1}^{i-1} p^j_{l,l+1} \prod_{k=1}^{j-1} (1-P_k), \tag{3.29}$$

where for any k that corresponds to a calibration period $P_k = 0$.

3.8.3. Derivation of the Stationary Distribution for Each Service Model

Single EPR Pair Generation with Strict Resource Reservation

Theorem 11. *The stationary distribution $\pi(\mathbf{x})$ of the system with single EPR pair generation while in strict resource reservation mode is given by*

$$\pi(\mathbf{x}) = \left\{ \left(\sum_{\mathbf{y}\in\mathscr{S}} \prod_{f\in\mathscr{F}} \prod_{i=1}^{L^f} \frac{(\rho^f_i)^{y^f_i}}{y^f_i!} \right)^{-1} \right\} \prod_{f\in\mathscr{F}} \prod_{i=1}^{L^f} \frac{(\rho^f_i)^{x^f_i}}{x^f_i!}, \tag{3.30}$$

where ρ^f_i is the traffic intensity of the i^{th} phase of a session corresponding to flow f.

Proof. To obtain the stationary probability $\pi(\mathbf{x})$ of the system with state space $\mathscr{S}$ as defined in (3.20), we apply the local balance approach. Henceforth, we adopt the convention that $\pi(\mathbf{x}) = 0$ for any $\mathbf{x} \notin \mathscr{S}$. Consider any state $\mathbf{x} \in \mathscr{S}$; we have that

- The rate of leaving $\mathbf{x}$ due to outside arrival coming into the system is given by

$$A = \pi(\mathbf{x}) \sum_{f\in\mathscr{F}} \nu^f_1(\mathbf{x}); \tag{3.31}$$

- The rate of entering $\mathbf{x}$ due to job departure to outside of the system is given by

$$A' = \sum_{f\in\mathscr{F}} \sum_{i=1}^{L^f} \pi(\mathbf{x}+\mathbf{e}^f_i)(x^f_i+1)p^f_i\mu^f_i, \tag{3.32}$$

where we take advantage of our convention that $\pi(\mathbf{x}+\mathbf{e}_i^f)=0$ if $\mathbf{x}+\mathbf{e}_i^f \notin \mathscr{S}$, as well as use the fact that $p_i^f=0$ if i corresponds to a calibration phase of a session belonging to flow f.

- The rate of leaving $\mathbf{x}$ due to departure from phase i of flow f is given by

$$B_i^f = \pi(\mathbf{x}) x_i^f \mu_i^f; \tag{3.33}$$

- The rate of entering $\mathbf{x}$ due to an arrival at phase i of flow f is given by

$$B_i'^f = \pi(\mathbf{x}-\mathbf{e}_i^f)\nu_i^f(\mathbf{x}-\mathbf{e}_i^f) + \sum_{j\neq i}\pi(\mathbf{x}+\mathbf{e}_j^f-\mathbf{e}_i^f)(x_j^f+1)p_{j,i}^f\mu_j^f. \tag{3.34}$$

To obtain the stationary distribution of the system, we solve the following equations and then show that this solution is in fact the stationary distribution of the system,

$$\pi(\mathbf{x}+\mathbf{e}_i^f) = \frac{\lambda_i^f(\mathbf{x})}{\mu_i^f(x_i^f+1)}\pi(\mathbf{x}) \tag{3.35}$$

$$= \begin{cases} \frac{\rho_i^f}{x_i^f+1}\pi(\mathbf{x}), & \text{if } \mathbf{x}+\mathbf{e}_i^f \in \mathscr{S}, \\ 0, & \text{else} \end{cases} \tag{3.36}$$

where $\rho_i^f \equiv \lambda_i^f/\mu_i^f$. Substituting this expression into $A=A'$ and simplifying, we obtain

$$\sum_{f\in\mathscr{F}}\nu_1^f(\mathbf{x}) = \sum_{f\in\mathscr{F}}\sum_{i=1}^{L^f}\lambda_i^f(\mathbf{x})p_i^f. \tag{3.37}$$

The expression above is the traffic conservation equation: the aggregate arrival rate into the system while in state $\mathbf{x}$ equals the overall departure rate from it. Continuing the application of local balance, we require that for all i and f, $B_i^f = B_i'^f$. Note that if $\mathbf{x}=\mathbf{0}$, then both B_i^f and $B_i'^f$ are zero, since we cannot leave this state due to a departure from any phase, and we cannot enter this state due to an arrival to any phase, respectively, since there are no active sessions while in state $\mathbf{0}$. For any other $\mathbf{x}\in\mathscr{S}$, we require

$$\pi(\mathbf{x})x_i^f\mu_i^f = \pi(\mathbf{x}-\mathbf{e}_i^f)\nu_i^f(\mathbf{x}-\mathbf{e}_i^f) + \sum_{j\neq i}\pi(\mathbf{x}+\mathbf{e}_j^f-\mathbf{e}_i^f)(x_j^f+1)p_{j,i}^f\mu_j^f. \tag{3.38}$$

Using (3.36) with (3.38) and simplifying, we obtain

$$\pi(\mathbf{x}-\mathbf{e}_i^f)\lambda_i^f(\mathbf{x}-\mathbf{e}_i^f) = \pi(\mathbf{x}-\mathbf{e}_i^f)\nu_i^f(\mathbf{x}-\mathbf{e}_i^f) + \sum_{j\neq i}\pi(\mathbf{x}-\mathbf{e}_i^f)\lambda_j^f(\mathbf{x}-\mathbf{e}_i^f)p_{j,i}^f. \tag{3.39}$$

If $\mathbf{x}-\mathbf{e}_i^f \notin \mathscr{S}$, then $\pi(\mathbf{x}-\mathbf{e}_i^f)=0$ and the equation above holds; else we obtain

$$\lambda_i^f(\mathbf{x}-\mathbf{e}_i^f) = \nu_i^f(\mathbf{x}-\mathbf{e}_i^f) + \sum_{j\neq i}\lambda_j^f(\mathbf{x}-\mathbf{e}_i^f)p_{j,i}^f. \tag{3.40}$$

We have thus recovered the traffic equations for each individual phase of the sessions in the network: the total arrival rate into phase i of flow f equals the sum of the exogenous arrival rates and the arrival rates from other phases. Thus our solution satisfying (3.36) represents the stationary distribution.

Next, from (3.36), it follows that

$$\pi(\mathbf{x}) = D \prod_{f\in\mathscr{F}} \prod_{i=1}^{L^f} \frac{(\rho_i^f)^{x_i^f}}{x_i^f!}, \tag{3.41}$$

where $D = \pi(\mathbf{0})$. Using the fact that $\sum_{\mathbf{x}\in\mathscr{S}} \pi(\mathbf{x}) = 1$, we can solve for D, which results in the stationary distribution (3.30), where the term inside the curly brackets is $\pi(\mathbf{0})$. Note from (3.30) that if a subset of flows $\mathscr{F}' \subseteq \mathscr{F}$ never wish to generate requests – i.e., $\nu_1^f(\mathbf{x}) = 0, \forall \mathbf{x} \in \mathscr{S}, f \in \mathscr{F}'$ – then these flows' traffic intensities are equal to 0. If we take the convention that $0^0 \equiv 1$, then (3.30) supports this setting by considering only flows from $\mathscr{F} \setminus \mathscr{F}'$ when $\mathbf{x} \in \mathscr{S}$, and causing $\pi(\mathbf{x})$ to equal zero whenever $x_i^f \neq 0$ for $f \in \mathscr{F}'$. □

MULTIPLE EPR PAIR GENERATION WITH STRICT RESOURCE RESERVATION
In this service model the stationary distribution is given by (3.30). The same derivation as in Appendix 3.8.3 applies, with the application of the considerations in Appendix 3.8.1, which only affect the traffic intensities.

MULTIPLE EPR PAIR GENERATION WITH RESOURCE RELINQUISHMENT
We derive the stationary distribution $\pi(\mathbf{x}), \mathbf{x} \in \mathscr{S}''$ for the service model with resource relinquishment and jump-over blocking as the retrial mechanism and show that the blocking probability is given by (3.13). The admissible state space $\mathscr{S}''$ is defined in Appendix 3.8.1 and the transitions possible in this service model are detailed in Appendix 3.8.1.

To obtain the stationary distribution of this system, we apply the local balance approach as in Appendix 3.8.3. The rate of leaving state $\mathbf{x} \in \mathscr{S}''$ due to an outside arrival is

$$A = \pi(\mathbf{x}) \sum_{f\in\mathscr{F}} \nu_1^f(\mathbf{x}), \tag{3.42}$$

and the rate of entering $\mathbf{x}$ due to termination of a session is

$$A' = \sum_{f\in\mathscr{F}} \sum_{i=1}^{L^f} \pi(\mathbf{x}+\mathbf{e}_i^f)(x_i^f+1)\mu_i^f p_i^f(\mathbf{x}+\mathbf{e}_i^f). \tag{3.43}$$

In (3.43), we can replace $p_i^f(\mathbf{x}+\mathbf{e}_i^f)$ with p_i^f since if $\mathbf{x}+\mathbf{e}_i^f \notin \mathscr{S}''$, then by our convention $\pi(\mathbf{x}+\mathbf{e}_i^f) = 0$. Recall that if $\mathbf{x}+\mathbf{e}_i^f \in \mathscr{S}''$ but i corresponds to any phase that does not belong to the last period of a session, then $p_i^f = 0$. Then, $A = A'$ yields a traffic conservation equation of the same form as (3.37), if we let $\pi(\mathbf{x})$ take the same form as (3.36) (note that the definition of $\lambda_i^f(\mathbf{x})$ is now given by (3.25) throughout). Local balance for each phase i of a session belonging to flow f yields the expected traffic equations

$$\lambda_i^f(\mathbf{x}) = \nu_i^f(\mathbf{x}) + \sum_{j\neq i} \lambda_j^f(\mathbf{x}) p_{j,i}^f(\mathbf{x}+\mathbf{e}_j^f-\mathbf{e}_i^f). \tag{3.44}$$

3.8.4. PROOFS

BLOCKING PROBABILITY FOR THE SINGLE EPR PAIR GENERATION WITH STRICT RESOURCE RESERVATION SERVICE MODEL

In this appendix we present a proof of Theorem 8, stated in Section 3.5.1.

Proof of Theorem 8. To begin, let $\mathrm{P}(\mathbf{q})$ denote the probability that EGS resources are occupied according to $\mathbf{q}$. Consider the following two events: $\Omega_1(h)$ is the event that h EGS resources are occupied, and $\Omega_2(i)$ is the event that flow f_i has available communication qubits. By the PASTA (Poisson Arrivals See Time Averages) property, we can write the probability that an arriving request of flow f_i sees h occupied resources (conditioned on the flow having enough qubits to generate a request), i.e., $\mathrm{P}(\Omega_1(h)|\Omega_2(i))$, as

$$\begin{aligned}\overline{\pi}_i(h) &= \frac{\mathrm{P}(\Omega_1(h)\cap\Omega_2(i))}{\mathrm{P}(\Omega_2(i))} \\ &= \left(\sum_{\mathbf{q}\in\mathscr{Q}'(i)}\mathrm{P}(\mathbf{q})\right)^{-1}\sum_{\mathbf{q}\in\mathscr{Q}(h)\cap\mathscr{Q}'(i)}\mathrm{P}(\mathbf{q}).\end{aligned} \tag{3.45}$$

Here we use the fact that flow f_i would not be able to initiate a session if any node associated with f_i does not have an unoccupied communication qubit.

The probability that flows occupy EGS resources according to $\mathbf{q}$ is

$$\begin{aligned}\mathrm{P}(\mathbf{q}) &= \sum_{\mathbf{x}:\mathbf{q}(\mathbf{x})=\mathbf{q}}\pi(\mathbf{x}) \\ &= \sum_{\mathbf{x}^{f_1}:q^{f_1}(\mathbf{x}^{f_1})=q_1}\cdots\sum_{\mathbf{x}^{f_F}:q^{f_F}(\mathbf{x}^{f_F})=q_F} D\prod_{f\in\mathscr{F}}\prod_{i=1}^{L^f}\frac{(\rho_i^f)^{x_i^f}}{x_i^f!},\end{aligned} \tag{3.46}$$

where $\mathbf{q}(\mathbf{x})$ is a vector containing $q^f(\mathbf{x}^f)$ values for $f\in\mathscr{F}$.

Recursive application of the multinomial theorem on (3.46) results in

$$P(\mathbf{q}) = D\prod_{j=1}^{F}\frac{1}{q_j!}\left(\sum_{i=1}^{L^{f_j}}\rho_i^{f_j}\right)^{q_j} \equiv D\prod_{j=1}^{F}\frac{(\rho^{f_j})^{q_j}}{q_j!}, \tag{3.47}$$

where $\rho^{f_j} := \sum_{i=1}^{L^{f_j}}\rho_i^{f_j}$. The constant D can be simplified similarly, after the additional step of rewriting the sum over $\mathbf{y}\in\mathscr{S}$ in terms of the total number of occupied EGS resources. Although D cancels out within the blocking probability, we present it in simplified form ($\tilde{D}$) for completeness:

$$\tilde{D}^{-1} = \sum_{\mathbf{y}\in\mathscr{S}}\prod_{f\in\mathscr{F}}\prod_{i=1}^{L^f}\frac{(\rho_i^f)^{y_i^f}}{y_i^f!} = \sum_{h=0}^{C}\sum_{\mathbf{q}\in\mathscr{Q}(h)}\prod_{j=1}^{F}\frac{(\rho^{f_j})^{q_j}}{q_j!}. \tag{3.48}$$

Using (3.47) and (3.45), we can obtain the blocking probabilities for the system – that is, the probability that an arriving request belonging to flow f_i sees C resources occupied

is given by

$$\overline{\pi}_i(C) = \left(\sum_{\mathbf{q} \in \mathscr{Q}'(i)} \prod_{j=1}^{F} \frac{\left(\rho^{f_j}\right)^{q_j}}{q_j!} \right)^{-1} \sum_{\mathbf{q} \in \mathscr{Q}(C) \cap \mathscr{Q}'(i)} \prod_{j=1}^{F} \frac{\left(\rho^{f_j}\right)^{q_j}}{q_j!}, \tag{3.49}$$

In the above equation the denominator term represents the probability that user-pairs of flow f_i have unoccupied communication qubits. Note that flow f_i triggers sessions only when both of its users have unoccupied communication qubits. □

Remark 1. *For our model the blocking probability for a given flow f is expressed as a conditional probability, as entanglement requests are triggered only when all the associated nodes have sufficient communication qubits. In [29], conditioning is not necessary since flows always trigger requests according to an unrestricted Poisson process. Further, in our model, blocking probabilities can be different for different flows as users may have different number of communication qubits whereas all the flows have the same blocking probabilities in [29].*

Remark 2. *Let us take a closer look at the form of ρ^f, for a given $f \in \mathscr{F}$ and validate that it is consistent with (3.3), the definition of the overall traffic intensity of flow f:*

$$\rho^f = \sum_{i=1}^{L^f} \rho_i^f = \sum_{i=1}^{L^f} \frac{\lambda_i^f}{\mu_i^f} = \sum_{i=1}^{L^f} \frac{\nu_1^f \tilde{p}_i^f}{\mu_i^f} = \nu_1^f \sum_{i=1}^{L^f} \frac{\tilde{p}_i^f}{\mu_i^f}. \tag{3.50}$$

Noting that the sum represents the mean duration of a type f session, we see that we recover (3.3), $\rho^f = \nu_1^f \times \mathbb{E}$[session duration].

Insensitivity Theorem for Single EPR Pair Generation with Strict Resource Reservation

In this appendix, we present a proof of Theorem 9, stated in Section 3.5.1.

Proof of Theorem 9. To show insensitivity to the distributions of periods we will prove that the stationary distribution for the total number of ongoing sessions of flows remains the same when attempts and calibration periods have either Coxian or exponential distributions with the same mean.

We first derive the expressions for the stationary distribution for the case where each attempt and calibration period are exponentially distributed. If M_A^f and M_C^f are the number of attempt and calibration periods, respectively, for flow f (and there are no idle periods in this system), then $M^f \equiv M_A^f + M_C^f$ is the total number of periods in each session of type f. The state of the system can then be described using the vector

$$\mathbf{Z} = \left[\mathbf{Z}^{f_1}, \ldots, \mathbf{Z}^{f_F}\right] = \left[Z_i^f, 1 \le i \le M^f, f \in \mathscr{F}\right], \tag{3.51}$$

where Z_i^f indicates the total number of ongoing sessions of type f in period i. Let L be the state dimension, i.e., $L \equiv \sum_{f \in \mathscr{F}} M^f$; then the admissible state space for this system is

given by

$$\mathscr{S}' = \left\{ \mathbf{Z} \in \mathbb{N}^L : \sum_{f \in \mathscr{F}} \sum_{i=1}^{M^f} Z_i^f \leq C, \sum_{f \in \mathscr{F}: n_k \in f} \sum_{i=1}^{M^f} Z_i^f \leq c_k, \forall\, k \in \{1, \ldots, K\} \right\}. \tag{3.52}$$

Let $1/\theta_j^f$ and $1/\sigma_j^f$ be the average duration of attempt and calibration periods j, respectively, for $f \in \mathscr{F}$. Further, let ω_j^f be the arrival rate into the j^{th} period of a session belonging to flow f. By the results of the previous subsection, we know that the stationary distribution for this system is

$$\pi'(\mathbf{Z}) = \left(\sum_{\mathbf{Y} \in \mathscr{S}'} \prod_{f \in \mathscr{F}} \prod_{i=1}^{M^f} \frac{(\xi_i^f)^{Y_i^f}}{Y_i^f!} \right)^{-1} \prod_{f \in \mathscr{F}} \prod_{j=1}^{M^f} \frac{(\xi_j^f)^{Z_j^f}}{Z_j^f!}, \tag{3.53}$$

where period j's traffic intensity for flow f is $\xi_j^f = \omega_j^f/\theta_j^f$ if j corresponds to an attempt period, and $\xi_j^f = \omega_j^f/\sigma_j^f$ if it corresponds to a calibration period. Analogous to the Coxian case, $\omega_j^f = \nu_j^f$ if $j = 1$, else it is

$$\omega_j^f = \nu_1^f \zeta_{1,2}^f \cdots \zeta_{j-1,j}^f, \tag{3.54}$$

where $\zeta_{l-1,l}^f$ is the probability of transitioning from period $l-1$ to period l of a session belonging to flow f. Let ζ_j^f be the probability of leaving the queueing network after the j^{th} queue due to a successful BSM at the EGS for creating an entanglement, and note that in this service mode $\zeta_j^f = 0$ if j corresponds to a calibration period.

To prove insensitivity, we assume that the average duration of a period in the exponential scenario is equal to the average duration of the corresponding period in the Coxian scenario. In other words, we have that for the j^{th} period, depending on whether it is an attempt or calibration period, respectively,

$$\frac{1}{\theta_j^f} = \sum_{i=1}^{N_A^f} \frac{r_{i,j}^f}{\mu_{i,j}^f}, \quad \text{or} \quad \frac{1}{\sigma_j^f} = \sum_{i=1}^{N_C^f} \frac{r_{i,j}^f}{\mu_{i,j}^f}, \tag{3.55}$$

where $r_{i,j}^f$ denotes the probability of reaching the i^{th} phase of the j^{th} period starting from its initial phase, and $1/\mu_{i,j}^f$ denotes the average duration of the i^{th} phase of period j within a flow-f session.

We further assume that for each attempt period of the exponential scenario, the entanglement success probability equals that of the corresponding attempt period in the Coxian scenario. That is, $\zeta_j^f = P_j^f = p_{\text{gen}}$ in this case, where P_j^f is the probability of leaving the queueing network during the j^{th} period of a flow-f session in the Coxian scenario, as computed in (3.28). This assumption is physically motivated for scenarios where the mean duration of an attempt is significantly shorter than the timescale of parameter drift

affecting a quantum node. For a detailed justification, see the discussion on success probability, Section 3.3.1.

In the Coxian scenario, we can rewrite the state representation in (3.17) as follows:

$$\mathbf{x} = [\mathbf{x}_1^{f_1}, \ldots, \mathbf{x}_{M^{f_1}}^{f_1}, \ldots, \mathbf{x}_1^{f_F}, \ldots, \mathbf{x}_{M^{f_F}}^{f_F}], \tag{3.56}$$

where M^{f_i} is the number of periods in sessions of type f_i. For any $\mathbf{x} \in \mathscr{S}$, let

$$q_j^f(\mathbf{x}_j^f) = \sum_i x_{i,j}^f,$$

i.e., this is the number of sessions in the j^{th} period of sessions belonging to flow f. Then for a given $\mathbf{Z} \in \mathscr{S}'$,

$$\sum_{\mathbf{x}: q_m^f(\mathbf{x}_m^f) = Z_m^f, \forall m, f} \pi(\mathbf{x}) = D \cdot \Bigg(\sum_{\mathbf{x}_1^{f_1}: q_1^{f_1}(\mathbf{x}_1^{f_1}) = Z_1^{f_1}} \cdots \sum_{\mathbf{x}_{M^{f_F}}^{f_F}: q_{M^{f_F}}^{f_F}(\mathbf{x}_{M^{f_F}}^{f_F}) = Z_{M^{f_F}}^{f_F}} \prod_{f \in \mathscr{F}} \prod_{j=1}^{M^f} \prod_{i=1}^{N_j^f} \frac{\left(\rho_{i,j}^f\right)^{x_{i,j}^f}}{x_{i,j}^f!} \Bigg), \tag{3.57}$$

where D is the normalizing constant of the Coxian distribution (see (3.30)), N_j^f is the number of phases in the j^{th} period of a session belonging to flow f, and $\rho_{i,j}^f = \lambda_{i,j}^f / \mu_{i,j}^f$ is the traffic intensity in the i^{th} phase of this period. Multiple applications of the multinomial theorem yield

$$\sum_{\mathbf{x}: q_m^f(\mathbf{x}_m^f) = Z_m^f, \forall m, f} \pi(\mathbf{x}) = D \prod_{f \in \mathscr{F}} \prod_{j=1}^{M^f} \frac{1}{Z_j^f!} \left(\sum_{i=1}^{N_j^f} \rho_{i,j}^f \right)^{Z_j^f}. \tag{3.58}$$

When j corresponds to an attempt period, $N_j^f = N_A^f$, and we have that

$$\sum_{i=1}^{N_j^f} \rho_{i,j}^f = \sum_{i=1}^{N_A^f} \rho_{i,j}^f = \sum_{i=1}^{N_A^f} \frac{\lambda_{i,j}^f}{\mu_{i,j}^f} \tag{3.59}$$

$$= \nu_1^f \sum_{i=1}^{N_A^f} \frac{\tilde{p}_{i,j}^f}{\mu_{i,j}^f} \tag{3.60}$$

$$= \nu_1^f \sum_{i=1}^{N_A^f} \frac{\prod_{l=1}^{i-1} p_{l,l+1}^{j,f} \prod_{k=1}^{j-1} \left(1 - P_k^f\right)}{\mu_{i,j}^f} \tag{3.61}$$

$$= \nu_1^f \prod_{k=1}^{j-1} \left(1 - P_k^f\right) \sum_{i=1}^{N_A^f} \frac{r_{i,j}^f}{\mu_{i,j}^f} \tag{3.62}$$

$$= \frac{\nu_1^f}{\theta_j^f} \prod_{k=1}^{j-1} \left(1 - \zeta_k^f\right) \tag{3.63}$$

$$= \frac{\nu_1^f}{\theta_j^f} \prod_{k=1}^{j-1} \zeta_{k,k+1}^f = \xi_j^f, \tag{3.64}$$

where in the last equality of (3.60) we use (3.24), in (3.61) we use (3.29); for (3.62) we use the fact that $\prod_{l=1}^{i-1} p_{l,l+1}^{j,f}$ simply represents the probability of going from the first phase of the period to the i^{th}; (3.63) follows from our two assumptions on the entanglement generation success probability and mean period duration; and the last equality of (3.64) follows from (3.54) and the definition of traffic intensity. We can use a similar argument to show that when j corresponds to a calibration period, the sum within (3.58) equals $\xi_j^f = \omega_j^f / \sigma_j^f$.

We next address the normalization constant D within (3.58):

$$\begin{aligned} D &= \sum_{\mathbf{y} \in \mathscr{S}} \prod_{f \in \mathscr{F}} \prod_{i=1}^{L^f} \frac{(\rho_i^f)^{y_i^f}}{y_i^f!} \\ &= \sum_{\mathbf{Z} \in \mathscr{S}'} \sum_{\mathbf{x}: q_m^{f'}(\mathbf{x}_m^{f'}) = Z_m^{f'}, \forall m, f'} \prod_{f \in \mathscr{F}} \prod_{j=1}^{M^f} \prod_{i=1}^{N_j^f} \frac{(\rho_{i,j}^f)^{y_{i,j}^f}}{y_{i,j}^f!}, \end{aligned} \tag{3.65}$$

where we have partitioned the state space $\mathscr{S}$ based on the number of jobs in each period of a session. Applications of the multinomial theorem as in (3.57), followed by utilization of (3.55), results in D's equivalence to the normalization constant of the exponential-scenario stationary distribution, see (3.53). This result, combined with (3.58), means that

$$\sum_{\mathbf{x}: q_m^f(\mathbf{x}_m^f) = Z_m^f, \forall m, f} \pi(\mathbf{x}) = \pi'(\mathbf{Z}). \tag{3.66}$$

The above equation shows that the stationary distribution for the total number of ongoing sessions of flows remains the same when attempts and calibration periods have either Coxian or exponential distributions with the same mean. From this, we conclude that the insensitivity property holds. □

BLOCKING PROBABILITY FOR THE MULTIPLE EPR PAIR GENERATION WITH RESOURCE RELINQUISHMENT SERVICE MODEL

In this appendix we provide a proof of Theorem 10. The proof relies on the phase-level system model introduced in Appendix 3.8.1 and the derivation of the stationary distribution of the system in this service model given in Appendix 3.8.3.

Proof of Theorem 10. In order to obtain the blocking probability, we must consider not only the external arrival process, but also the internal jump-over blocking mechanism. For the former, we can apply the Poisson-Arrivals-See-Time-Averages (PASTA) property as sessions arrive according to a Poisson process. For the latter, we give the corresponding result in Corollary 2 below that extends the result of "Departures See Time Averages"

stated in [29, Corollary 1], which was proven for a slightly different model presented in that manuscript – namely, there each active period is followed by an idle period and a flow generates sessions without any restrictions from users. In our case, both the users of a flow must have one unoccupied communication qubit to submit their request for EGS resource access. Corollary 2 enables us to show that the blocking probability of the first attempt of the session is identical to the blocking probability of its retrials.

Corollary 2. *For the service model introduced in Section 3.5.3, let* $\mathbf{x}+\mathbf{e}_i^f \in \mathscr{S}''$*, with* i *corresponding to an idle or calibration phase of the flow* f*, where* $\mathscr{S}''$ *is defined in Appendix 3.8.1. The probability that a session of flow* f *departing phase* i *from state* $\mathbf{x}+\mathbf{e}_i^f \in \mathscr{S}''$ *sees the system in state* $\mathbf{x} \in \mathscr{S}''$*, equals the stationary conditional probability that the system is in state* $\mathbf{x}$ *given that both nodes of the flow* f *have at least one unoccupied communication qubit.*

Proof. Define $\pi^{i,f}(\mathbf{x})$ as the probability that sessions belonging to flow f leaving an idle phase i see the network in state $\mathbf{x}$ immediately after their departure in the stationary regime. This probability is given by

$$\pi^{i,f}(\mathbf{x}) = \frac{\pi(\mathbf{x}+\mathbf{e}_i^f)\mu_i^f(x_i^f+1)}{\sum\limits_{\mathbf{y}:\mathbf{y}+\mathbf{e}_i^f \in \mathscr{S}''} \pi(\mathbf{y}+\mathbf{e}_i^f)\mu_i^f(y_i^f+1)}, \tag{3.67}$$

where the numerator represents the transition rate out of state $\mathbf{x}+\mathbf{e}_i^f$ and the denominator represents the transition rate out of all possible states associated with termination of sessions.

For $\mathbf{x}+\mathbf{e}_i^f \in \mathscr{S}''$ (and therefore, by the convexity of the state space, $\mathbf{x} \in \mathscr{S}''$), we know from the stationary distribution analysis that

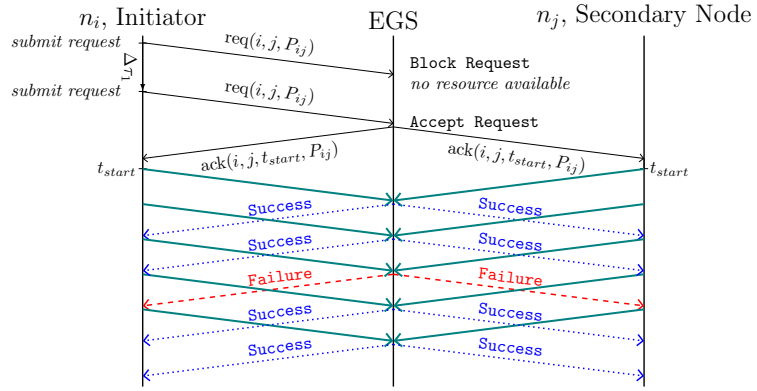
$$\pi(\mathbf{x}+\mathbf{e}_i^f) = \frac{\rho_i^f}{x_i^f+1}\pi(\mathbf{x}). \tag{3.68}$$

Using this result with (3.67) and (3.25), upon simplifying we obtain

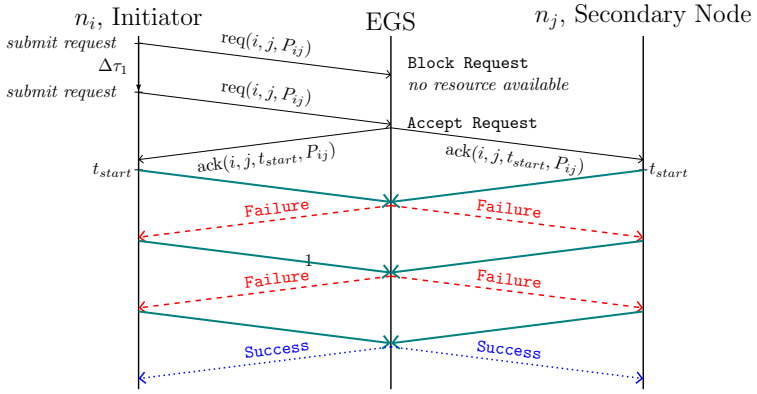
$$\pi^{i,f}(\mathbf{x}) = \frac{\pi(\mathbf{x})}{\sum\limits_{\mathbf{y}:\mathbf{y}+\mathbf{e}_i^f \in \mathscr{S}''} \pi(\mathbf{y})}. \tag{3.69}$$

□

From Corollary 2 and (3.69), the probability that a job of flow f leaving an idle period finds all EGS rource modules to be occupied is equal to $\frac{\sum_{\mathbf{x}:\mathbf{x}\in\mathscr{Q}(C)\cap\mathscr{Q}'(f)}\pi(\mathbf{x})}{\sum\limits_{\mathbf{y}:\mathbf{y}\in\mathscr{Q}'(f)}\pi(\mathbf{y})}$, which coincides with $\overline{\pi}_f(C)$. Therefore we conclude that for an active period that immediately follows an idle period, the blocking probability is the same as that for the first period of a session (recall that by our convention, the first period is always an active period). □



(a) CI protocol



(b) HE protocol

Figure 3.18: Communication sequences for the (a) CI and (b) HE protocols. The request field P_{ij} represents a packet of parameters, such as the number of attempts requested. $\Delta\tau_1$ is an exponentially distributed inter-arrival time between subsequent requests. Black arrows indicate classical messages. Teal arrows indicate pulses of light at the single photon level sent over optical fiber. Blue, dotted (red, dashed) arrows indicate a success (failure) flag communicated over a direct connection such as optical fiber.

3.8.5. PHYSICAL EGS OPERATION PROTOCOLS

CI PROTOCOL

The goal of a node pair (n_i, n_j) running a CI protocol is to generate a data record of a qubit string at each of nodes n_i and n_j such that the data records of n_i and n_j have entanglement-like correlations. These data records can serve as raw key, and the CI protocol is a form of QKD [1, 2] that uses a BSA as a measurement device at a central midpoint, known as MDI-QKD [41, 49, 50]. The advantage of MDI-QKD over traditional QKD protocols is that it is more secure, as it removes the possibility of detector side-channel attacks. Detector side-channel attacks have been shown to be easy to implement and challenging to defend against [54–56], hence this is an important advantage. To support running the

CI generation protocol, nodes n_i and n_j require a photon source. This source can be a multi-photon emitter attenuated to the single-photon level, such as an attenuated laser pulse, or a true single photon source like a quantum communication qubit. The CI generation protocol's compatibility with multi-photon sources is advantageous. In the early stages of quantum network development, multi-photon sources are more cost-effective and easier to build/purchase and maintain compared to single photon sources [41].

In broad terms, the protocol consists of four main stages. First, nodes exchange calibration information to ensure that the photons emitted by their sources will be spectrally indistinguishable. Second, each node encodes a qubit state into an emission of the photon source. Third, the photons are sent to a BSA, at which a BSM is attempted between the two received photons, which if successful, projects the photons into a maximally entangled state. Fourth, if the BSM succeeds, the records of the qubits prepared by (n_i, n_j) share entanglement-like correlations and a success flag is communicated to the nodes. In case of success the nodes store the qubit record. If the measurement is not successful, a failure flag is passively communicated to the nodes as the absence of sending a success flag; after the RTT time between each node and the EGS passes, plus some buffer time accounting for measurement and jitter, nodes discard qubit records from failed attempts. It is not necessary to wait for the outcome flag before commencing subsequent attempts. However, if the nodes use communication qubits to generate the single photons, the communication qubits must be measured in between subsequent attempts in order to obtain the data record. If a simple photon source is used this measurement step is not necessary, as the data record corresponds to the preparation record of the photon. Sequential execution of the second and third stages constitute a single CI generation attempt and may be repeated in batches. For any attempt, the fourth stage must eventually occur – the outcome flag must eventually be received by the nodes. Otherwise, a stop of the protocol may be triggered.

A communication sequence between the node pair (n_i, n_j) allocated use of an EGS resource to perform CI generation is illustrated in Figure 3.18a.

Comparison of the HE and CI Protocols

The main difference between these two protocols is that in the HE protocol a successful BSM at the EGS means that nodes become entangled whereas in the CI protocol a successful BSM at the EGS means that the data records corresponding to the photons sent to the EGS will display entanglement like correlations. Other important differences are the repetition rate and the probability that single attempts succeed. See Figures 3.18a and 3.18b for visualization of how the communication sequences differ between the protocols. The frequency at which CI generation can be attempted is inherently faster than HE generation because subsequent attempts do not require waiting for the outcome flag to be received (step four in the HE and CI communication sequences). The rate at which CI generation can be attempted with simple photon sources is limited by the emission rate of the photon source, the channel capacity of the fiber over which photons are sent, or the rate at which the BSA detectors become responsive again following detection of a photon (recovery from dead-time), whichever of these factors is most restrictive. If quantum communication qubits are used to enact the CI generation protocol, the other potential rate limiting factor is how quickly the qubits can be measured following photon emission.

To summarize, subsequent attempts of the CI generation protocol can begin before the completion of previous attempts, and the rate limiting factors in an implementation may allow for significantly high attempt repetition rates (MHz or GHz). In contrast, the HE generation protocol requires nodes to receive the heralding flag of any previous attempt before commencing any subsequent attempt, i.e., attempts must be non-overlapping. In a local quantum network with optical fiber links between nodes and the EGS ranging from 5 to 50 km, the RTTs vary from approximately 25 μs to 250 μs. Such RTTs dominate the time duration of a single HE generation attempt, limiting the maximum repetition rate of attempts in the protocol to the level of kHz.

A real world implementation of CI generation with simple photon sources [41] displayed a high total probability of a single attempt succeeding. In this implementation the main limitations against an attempt succeeding were loss of a photon traveling in fiber or a detection pattern incompatible with projection into a maximally entangled state. In contrast, in real world implementations of single slick bipartite heralded entanglement generation the probability of an attempt succeeding is limited by the probability of single photon emission from the communication qubit, and further suppressed by the probability of losing a photon traveling in fiber and the success probability of a BSM.

3.8.6. SIMULATION IMPLEMENTATION

Simulations of the discrete, Cox and exponential models allow a method of comparing the predictions of Theorems 8 and 9 to the blocking probabilities we would expect to observe in a real implementation where the distribution governing the duration of events coincides with the simulated model. The discrete simulation mode is implemented as a discrete time simulation where the time step is set to the duration of a single attempt, which is the shortest timescale in the system. The Poisson process by which flows submit requests is implemented by scaling the exponentially distributed inter-arrival times between requests to a number of time steps and rounding up to the nearest time step. The exponential and Cox simulation modes are implemented as discrete event simulations.

The expected number of requests placed over the duration of each simulation depends on the request arrival rate from each flow, ν_1^f. Each simulation corresponded to 1150.73 seconds of simulated real-time. The data in Figure 3.7 is representative for a discussion of the expected number of requests placed over that time. For the minimum (maximum) request arrival rate simulated, the expected number of requests from each flow is 100 (1135). Since there are 28 flows in total, the total expected number of requests placed during a simulation is then 2800 (31780) for the minimum (maximum) request rate simulated. Note that we in general observe very close agreement between the blocking probabilities resulting from discrete/exponential/Cox simulation types as well as with our numeric evaluation of (3.7). The magnitude of disagreement between simulation types may be referred to as the error between simulation types. To investigate whether these errors change systematically when simulation parameters are altered we have executed simulations with various time-step durations pertaining to the discrete simulation as well as with various numbers of attempts per batch of entanglement generation attempts. Under changes to these parameters, the errors between simulation types remain very small ($< 1\%$) and do not change in any systematic way.

In the implementation of the multiple EPR pair generation with resource relinquish-

Parameter type	Value
Number of phases	4
Mean duration of phase 1	$0.41\overline{6} \cdot T_{\text{attempt}}$
Mean duration of phase 2	$0.521 \cdot T_{\text{attempt}}$
Mean duration of phase 3	$0.651 \cdot T_{\text{attempt}}$
Mean duration of phase 4	$0.814 \cdot T_{\text{attempt}}$
Transition probabilities	(0.6, 0.48,0.384, 0.307)

Table 3.5: Cox distribution parameters for individual entanglement generation attempts in the homogeneous traffic scenario.

ment (jump-over) service mode, the mean duration of the idle periods, T_{idle}, is set to be equivalent to the mean duration of the calibration periods of the single/multiple EPR pair generation with strict resource reservation service modes, i.e., $T_{\text{idle}} = T_{\text{calib}}$. The motivation for this implementation choice is to enable direct comparison between the different service modes, as in Figure 3.9.

In the non-homogeneous traffic scenario considered, for every flow, if one or both of the users involved is connected by a longer link, then the attempt time for the flow is lengthened to accommodate the greater RTT between the more distant node and the EGS. Thus flows in S_2 require longer resource reservation times to achieve the same number of attempts. As mentioned above, in the discrete simulations of homogeneous traffic, the time step is set to the duration of a single attempt. In the implementation of our non-homogeneous traffic model, we set the time step in discrete simulations to the duration of an attempt in S_1, which is the shortest time scale in the system. We then fix the mean attempt time in S_2 to double the mean attempt time of flows in S_1. This time is longer than the total of the RTT required for flows in S_2 plus the buffer included to capture measurement times for flows in S_1. Hence we have modeled adding some additional buffer time to each attempt for flows in S_2. As a consequence, in discrete simulations the duration of each attempt for flows in S_2 is exactly two-time steps. We assume all flows submit requests at the same rate, thus isolating the source of non-homogeneity to the different attempt times in sets S_1 and S_2, which are largely due to the different link-lengths.

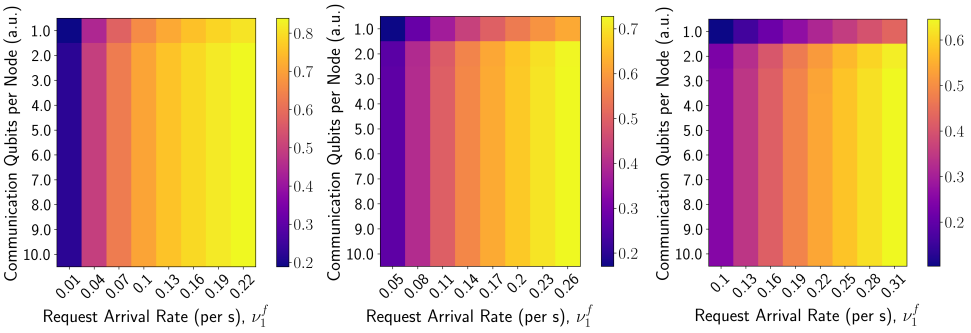
3.8.7. EXTENDED DATA

REFERENCES

[1] B. H. Charles and G. Brassard, *Quantum cryptography: Public key distribution and coin tossing,* Theoretical Computer Science **560**, 7 (2014).

[2] A. K. Ekert, *Quantum cryptography based on bell's theorem,* Phys. Rev. Lett. **67**, 661 (1991).

[3] P. Arright and L. Salvail, *Blind quantum computation,* International Journal of Quantum Information **04**, 883 (2006).



(a) one EGS Resource ($C = 1$) (b) two EGS Resources ($C = 2$) (c) three EGS Resources ($C = 3$)

Figure 3.19: Heatmaps of the average blocking probability per flow when the number of communication qubits per node and the request arrival rates are varied. Data results from numeric evaluation of (3.7) for an EGS with 20 nodes and serving $\binom{20}{2} = 190$ flows, one for each possible node pairing. Session traffic is homogeneous.

[4] A. Broadbent, J. Fitzsimons, and E. Kashefi, *Universal blind quantum computation,* in *2009 50th Annual IEEE Symposium on Foundations of Computer Science* (IEEE, Atlanta, Georgia, USA, 2009) pp. 517–526.

[5] D. Leichtle, L. Music, E. Kashefi, and H. Ollivier, *Verifying bqp computations on noisy devices with minimal overhead,* PRX Quantum **2**, 040302 (2021).

[6] D. Gottesman, T. Jennewein, and S. Croke, *Longer-baseline telescopes using quantum repeaters,* Phys. Rev. Lett. **109**, 070503 (2012).

[7] P. Kómár, E. M. Kessler, M. Bishof, L. Jiang, A. S. Sørensen, J. Ye, and M. D. Lukin, *A quantum network of clocks,* Nature Physics **10**, 582 (2014).

[8] V. Giovannetti, S. Lloyd, and L. Maccone, *Quantum-enhanced positioning and clock synchronization,* Nature **412**, 417 (2001).

[9] X. Guo, C. R. Breum, J. Borregaard, S. Izumi, M. V. Larsen, T. Gehring, M. Christandl, J. S. Neergaard-Nielsen, and U. L. Andersen, *Distributed quantum sensing in a continuous-variable entangled network,* Nature Physics **16**, 281 (2020).

[10] W. J. Munro, K. Azuma, K. Tamaki, and K. Nemoto, *Inside quantum repeaters,* IEEE Journal of Selected Topics in Quantum Electronics **21**, 78 (2015).

[11] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters, *Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels,* Physical review letters **70**, 1895 (1993).

[12] D. Bouwmeester, J.-W. Pan, K. Mattle, M. Eibl, H. Weinfurter, and A. Zeilinger, *Experimental quantum teleportation,* Nature **390**, 575 (1997).

[13] S. Gauthier, G. Vardoyan, and S. Wehner, *An architecture for control of entanglement generation switches in quantum networks,* IEEE Transactions on Quantum Engineering **4**, 1 (2023).

[14] S. L. Braunstein and A. Mann, *Measurement of the Bell operator and quantum teleportation,* Physical Review A **51**, R1727 (1995).

[15] M. Michler, K. Mattle, H. Weinfurter, and A. Zeilinger, *Interferometric Bell-state analysis,* Physical Review A **53**, R1209 (1996).

[16] P. Walther and A. Zeilinger, *Experimental realization of a photonic Bell-state analyzer,* Physical Review A **72**, 010302 (2005).

[17] M. Koyama, C. Yun, A. Taherkhani, N. Benchasattabuse, B. O. Sane, M. Hajdušek, S. Nagayama, and R. V. Meter, *Optimal switching networks for paired-egress bell state analyzer pools,* in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, Vol. 01 (2024) pp. 1897–1907.

[18] R. A. Thompson, *Traffic capabilities of two rearrangeably nonblocking photonic switching modules,* AT&T Technical Journal **64**, 2331 (1985).

[19] J. Preskill, *Quantum computing in the nisq era and beyond,* Quantum **2**, 79 (2018).

[20] A. K. Erlang, *Solution of some problems in the theory of probabilities of significance in automatic telephone exchanges,* Post Office Electrical Engineer's Journal **10**, 189 (1917).

[21] M. Harchol-Balter, *Performance modeling and design of computer systems: queueing theory in action* (Cambridge University Press, 2013).

[22] A. Abane, M. Cubeddu, V. S. Mai, and A. Battou, *Entanglement routing in quantum networks: A comprehensive survey,* IEEE Transactions on Quantum Engineering **6**, 1 (2025).

[23] L. Bacciottini, M. G. D. Andrade, S. Pouryousef, E. A. Van Milligen, A. Chandra, N. K. Panigrahy, N. S. V. Rao, G. Vardoyan, and D. Towsley, *Leveraging internet principles to build a quantum network,* IEEE Network , 1 (2025).

[24] S. Pouryousef, H. Shapourian, and D. Towsley, *Analysis of asynchronous protocols for entanglement distribution in quantum networks,* (2024), arXiv:2405.02406 [quant-ph] .

[25] B. A. Sevast'yanov, *An ergodic theorem for Markov processes and its application to telephone systems with refusals,* Theory of Probability & Its Applications **2**, 104 (1957).

[26] T. Bonald, *Insensitive queueing models for communication networks,* in *Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools*, valuetools '06 (Association for Computing Machinery, New York, NY, USA, 2006) p. 57–es.

[27] R. Serfozo, *Introduction to Stochastic Networks* (Springer, New York, NY, USA, 1999).

[28] W. Whitt, *Continuity of generalized semi-markov processes,* Mathematics of Operations Research - MOR **5**, 494 (1980).

[29] T. Bonald, *The Erlang model with non-Poisson call arrivals,* ACM SIGMETRICS Performance Evaluation Review **34**, 276 (2006).

[30] F. Kelly, *Charging and rate control for elastic traffic,* European transactions on Telecommunications **8**, 33 (1997).

[31] G. Vardoyan, S. Guha, P. Nain, and D. Towsley, *On the Stochastic Analysis of a Quantum Entanglement Distribution Switch,* IEEE Transactions on Quantum Engineering **2**, 1 (2021).

[32] G. Vardoyan, S. Guha, P. Nain, and D. Towsley, *On the exact analysis of an idealized quantum switch,* Performance Evaluation **144**, 102141 (2020).

[33] 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 (IEEE, New York, NY, USA, 2023) pp. 1–10.

[34] W. Dai, A. Rinaldi, and D. Towsley, *The capacity region of entanglement switching: Stability and zero latency,* in *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)* (IEEE, Broomfield, CO, USA, 2022) pp. 389–399.

[35] T. Vasantam and D. Towsley, *A throughput optimal scheduling policy for a quantum switch,* in *Quantum Computing, Communication, and Simulation II*, Vol. 12015, edited by P. R. Hemmer and A. L. Migdall, International Society for Optics and Photonics (SPIE, San Francisco, CA, USA, 2022) p. 1201505.

[36] L.-M. Duan, M. D. Lukin, J. I. Cirac, and P. Zoller, *Long-distance quantum communication with atomic ensembles and linear optics,* Nature **414**, 413 (2001).

[37] C. Cabrillo, J. I. Cirac, P. García-Fernández, and P. Zoller, *Creation of entangled states of distant atoms by interference,* Phys. Rev. A **59**, 1025 (1999).

[38] 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. **98**, 190503 (2007).

[39] N. Sangouard, C. Simon, H. de Riedmatten, and N. Gisin, *Quantum repeaters based on atomic ensembles and linear optics,* Rev. Mod. Phys. **83**, 33 (2011).

[40] N. Sangouard, R. Dubessy, and C. Simon, *Quantum repeaters based on single trapped ions,* Phys. Rev. A **79**, 042340 (2009).

[41] 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 **5**, 186 (2022).

[42] 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 **558**, 268 (2018).

[43] 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 **497**, 86 (2013).

[44] 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 **3**, 538 (2007).

[45] 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. **130**, 050803 (2023).

[46] 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 **438**, 828 (2005).

[47] 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 **316**, 1316 (2007).

[48] 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 **607**, 69 (2022).

[49] H.-K. Lo, M. Curty, and B. Qi, *Measurement-device-independent quantum key distribution,* Phys. Rev. Lett. **108**, 130503 (2012).

[50] S. L. Braunstein and S. Pirandola, *Side-channel-free quantum key distribution,* Phys. Rev. Lett. **108**, 130502 (2012).

[51] 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 **372**, 259 (2021).

[52] C. H. Bennett, G. Brassard, S. Popescu, B. Schumacher, J. A. Smolin, and W. K. Wootters, *Purification of noisy entanglement and faithful teleportation via noisy channels,* Physical review letters **76**, 722 (1996).

[53] D. Deutsch, A. Ekert, R. Jozsa, C. Macchiavello, S. Popescu, and A. Sanpera, *Quantum privacy amplification and the security of quantum cryptography over noisy channels,* Physical review letters **77**, 2818 (1996).

[54] H. Lu, C.-H. F. Fung, and Q.-Y. Cai, *Two-way deterministic quantum key distribution against detector-side-channel attacks,* Phys. Rev. A **88**, 044302 (2013).

[55] V. Makarov, A. Anisimov, and J. Skaar, *Effects of detector efficiency mismatch on security of quantum cryptosystems,* Phys. Rev. A **74**, 022313 (2006).

[56] Y. Zhao, C.-H. F. Fung, B. Qi, C. Chen, and H.-K. Lo, *Quantum hacking: Experimental demonstration of time-shift attack against practical quantum-key-distribution systems,* Phys. Rev. A **78**, 042333 (2008).

4

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

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 realize 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 realize 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.

4.1. Introduction

The realization 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].

4.1.1. STRUCTURE

The rest of this paper is structured as follows: In 4.2 we give some background and related work and discuss how our work here differs. In 4.3 we discuss design considerations which inform the design of our architecture. In 4.4 we lay out our proposed network architecture. In 4.5 we give an example of an implementation of the architecture. Finally in 4.6 we evaluate the performance of our architecture using a specific implementation, and in 4.7 we comment on future directions for research. A table summarising all the notation we use in this paper can be found in Appendix 4.8.1.

4.2. BACKGROUND

4.2.1. 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 realized 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.

4.2.2. 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.

In [37], Monga *et al.* also propose a centrally controlled quantum network architecture. Whilst this architecture does mention real-time control of quantum end nodes called Q-Nodes, there is no explicit concept of scheduling the execution of arbitrary quantum network applications on end nodes. In contrast, our architecture is explicitly designed to be compatible with Qoala running on QNodeOS, which provides this functionality. We also incorporate features into our architecture to support local application scheduling, such as distributing network schedules early to allow time for local schedules to be constructed. Additionally, the architecture in [37] does not provide details about the format demands should be submitted in, nor the approach taken for allocating or scheduling the use of network resources. In contrast, we provide such a demand format, motivated by the requirements to be able to execute quantum network applications, as well as an

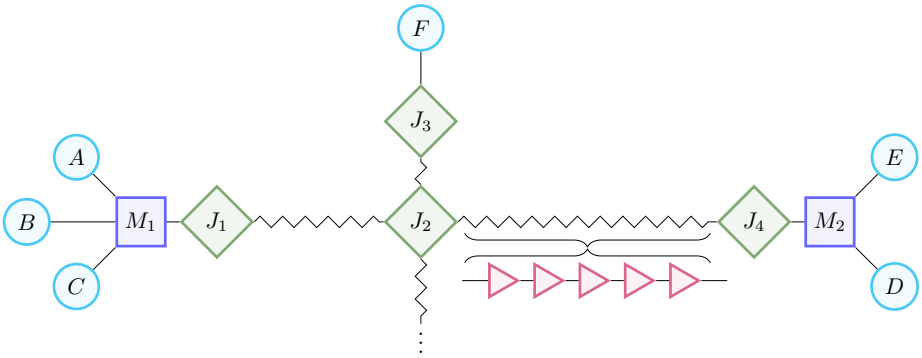


Figure 4.1: A general quantum network may be built from four kinds of devices: end nodes, metropolitan hubs, repeater chains, and junction nodes (see 4.3.1). These devices may be connected in the manner illustrated. Circles A-G represent user controlled end nodes, squares M_i are metropolitan hubs and diamonds J_i are junction nodes. Repeater chains are represented by zig-zag lines, while individual quantum repeater nodes are represented by triangles.

approach and method for producing network schedules allocating the use of shared network resources to pairs of end nodes.

There have also been several SDN architectures proposed for quantum key distribution (QKD), e.g. [38–47]. 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 [48]. Furthermore, we are focused on the execution of these applications rather than adding extra security to a classical network as in [38].

Our architecture also has similarities to a so-called *Time-triggered Ethernet (TTEthernet)* architecture [49]. 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 [50, 51], 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.

4.3. 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. Table 4.1 summarizes key parameters defined in our architecture and which entities provide these parameters in an implementation.

4.3.1. NETWORK

DEVICES AND COMPONENTS

We consider a quantum network comprised of four types of devices. Figure 4.1 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 [52–59], or a device which is capable of preparing/measuring single photons, such as in [60, 61]. 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 [62–64] and quantum routers [65], which employ quantum memories at the hub, or alternatively entanglement generation switches [66, 67], 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, 68–71]. We treat repeater chains as a single network component 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.

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.

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, 39]. 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 4.3.1). However, we do not assume it knows whether or not any given attempt to generate an entangled link succeeds or fails.

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 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 realizes 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, 72], on the repeater chain in order to create the required entangled links.

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, 68] (Figure 4.2). In particular this means that the nodes attempting to create an entangled link receive a success or failure outcome, indicating

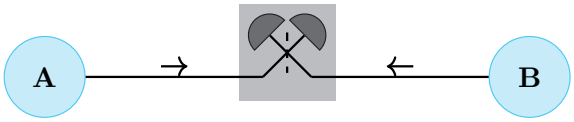


Figure 4.2: Example of an entanglement generation protocol (heralded entanglement). Nodes A and B probabilistically send photons to a 'heralding station' located midway between them. This heralding station consists of (single) photon detectors and a beamsplitter. Depending on the pattern of photons detected, 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].

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 4.2. 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 4.1 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.

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 [73] 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.

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.

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 [15, 53]. Precise timing of a sequence of operations is crucial for processes like entanglement generation [54, 55, 58, 59, 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.

4.3.2. 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.

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 realizing 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.

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]).

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, 54]. We do not make any assumptions about the classical processing capabilities of the end nodes.

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.

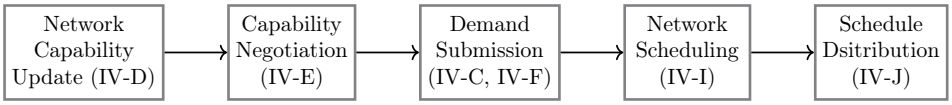


Figure 4.3: Flow of information in the architecture. The processes of 'Network Capability Update' and 'Capability Negotiation' allow the nodes to gather enough information to be able to submit a unified demand. These demands are then used to construct a central network schedule, which in turn is used when constructing the local node schedules.

4.4. 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 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 4.3. Example timings of the distinct processes are illustrated in Figure 4.4. A detailed overview of our architecture is illustrated by the interaction diagram in Figure 4.5.

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.

4.4.1. 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 4.4 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

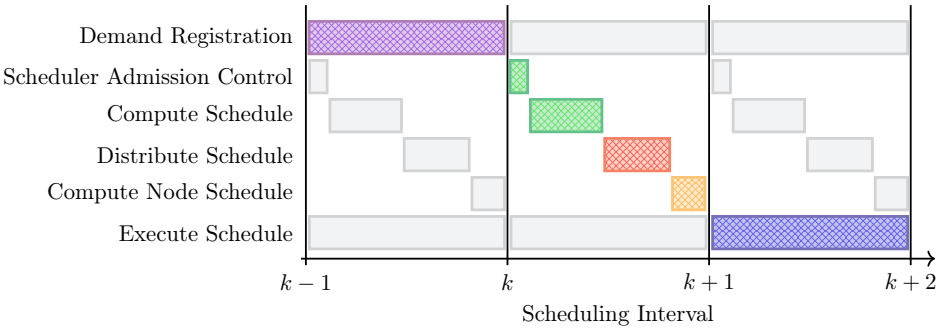


Figure 4.4: Example of the timings for computing, distributing and executing the kth network schedule (in colour and hatched). In grey and unhatched are the corresponding operations for preceding and subsequent network schedules. The process of registering received demands is continuous, however we indicate 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.

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.

4.4.2. 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 14 (Application Session)**.** *Suppose end nodes* $\mathcal{N} = (\texttt{node1}, \texttt{node2}, \ldots)$ *wish to execute application* App *at least* $N_{\texttt{inst}}$ *times, before time* $t_{\texttt{expiry}}$.

Then we write the corresponding application session as

$$\mathcal{S} = (\mathcal{N}, \texttt{App}, N_{\texttt{inst}}, t_{\texttt{expiry}}). \tag{4.1}$$

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 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 timescale of the network. For instance, if the network updates (see §4.4.9) 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.

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 15 (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 16 (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 §4.4.3) is generated, and a signal confirming this is passed to the local scheduler.*

Definition 17 (Successful Application Instance Execution)**.** *We say an application instance executed successfully if all blocks are successfully executed, including receiving confirmation signals.*

Definition 18 (Quantum Successful Execution)**.** *We say an application instance achieved quantum success if it was successfully executed and the correct outcome was obtained.*

Definition 19 (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.

4.4.3. 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 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.

Packets of Entanglement

We first address specification of the quality and quantity of entanglement required. Recall from §4.3.2 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 20 (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 21 (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 22 (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 $\mathscr{P}$ 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.

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 $N_{\texttt{inst}}$ 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*.

FULL DEMAND

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

$$\mathscr{D} = \left(\left\{ (w, s, F, R)_p \right\}_{p \in \mathscr{P}}; t_{\texttt{minsep}}, t_{\texttt{expiry}}; N_{\texttt{inst}} \right), \tag{4.2}$$

where $\mathscr{P}$ 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.

4.4.4. 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 4.5).

The application stack obtains this information in the following manner. Firstly, a query is sent to the quantum network agent (QNA) (see Figure 4.5). 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.

4.4.5. 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 4.5).

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.

4.4.6. DEMAND REGISTRATION

When demands are received by the central controller (interaction **C** in Figure 4.5), 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.

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.

4.4.7. SESSION INITIALISATION

Following capability negotiation and demand submission, each end node performs some initial configuration (interaction **D** in Figure 4.5). 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 [15, 16], ensuring that generated entanglement is assigned to the correct application.

4.4.8. 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.

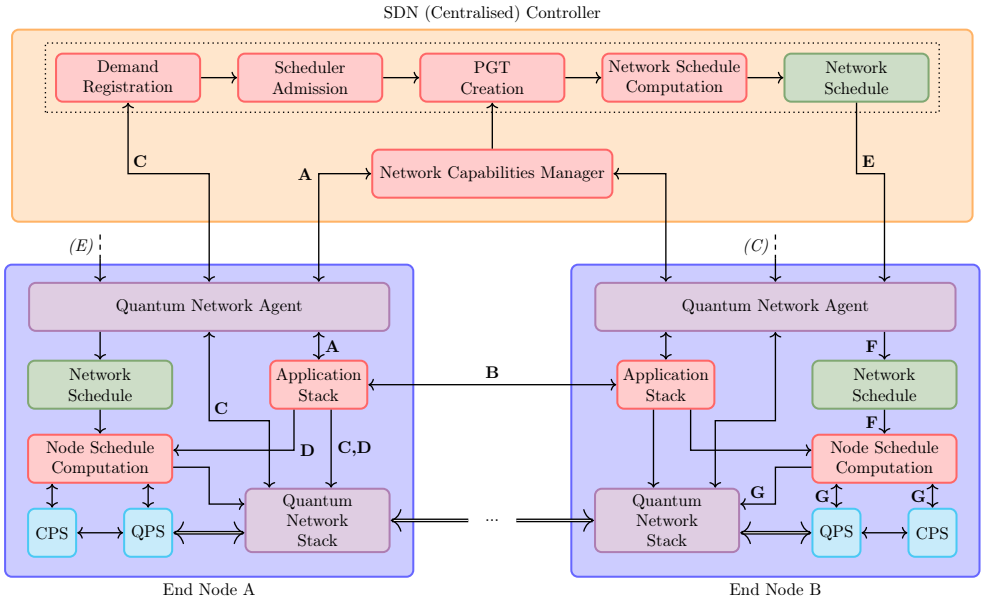


Figure 4.5: Interaction Diagram for our proposed quantum network architecture. Elements in red are local software components. Single stroke arrows represent purely classical interactions and double-stroke arrows represent quantum interactions. The dotted arrows denote the corresponding interaction from 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 (4.4.4). 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).

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

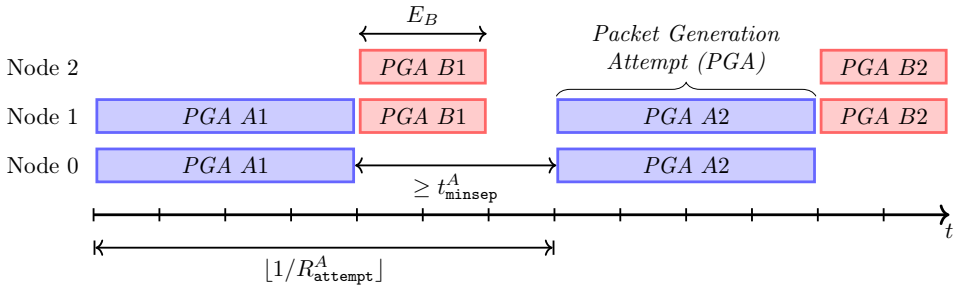


Figure 4.6: Excerpt from an example network schedule with two PGTs $\tau_A = ((4, 1/7, \{0, 1, M\}), 2, 100)$ and $\tau_B = ((2, 1/7, \{1, 2, M\}), 0, 100)$ on three nodes, where M represents a number of resources at a metropolitan hub.

with $p_{\texttt{packet}}$. We will see in §3 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.

Algorithm 2: Determining the length of PGAs

Input : w, s, p_{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_{succ}.

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 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, $\epsilon_{\text{service}}$, $p_{\texttt{packet}}$
Output : Minimum possible rate at which to schedule PGAs.

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

If permitted by the service model and any agreements 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.

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.

COMPLETE PACKET GENERATION TASK

It is now possible to define the complete PGT.

Definition 23 (Realization of a demand)**.** *A realization 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 realizations as $\mathscr{R}$.

Definition 24 (Packet Generation Task)**.** *A packet generation task (PGT) is the following tuple:*

$$\tau = \left(\left\{ (E, R_{\texttt{attempt}}, \rho)_r \right\}_{r \in \mathscr{R}},\ t_{\texttt{minsep}}, t_{\texttt{expiry}} \right) \tag{4.3}$$

where for each possible realization $r \in \mathscr{R}$:

- *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 4.6.

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 realize 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 $\mathscr{R}$ to a single realization/mode in which the PGT is always scheduled.

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

Table 4.1: Table of entities (sources) and the parameters they provide.

4.4.9. 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 4.4 illustrates possible timings for each of these processes.

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;

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 automatically 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, $\mathscr{U}_r$, should satisfy

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

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

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.

4.4.10. 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 4.4, 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 4.5). 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 4.5).

4.5. Implementation

In order to be able to perform an evaluation in §4.6, 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.

4.5.1. 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

$$T_{\texttt{attempt}} = \left\lfloor \frac{1}{R_{\texttt{attempt}}\, t_{\text{timeslot}}} \right\rceil_{\mathbb{N}}, \tag{4.4}$$

where we write $\lfloor x \rceil_{\mathbb{N}}$ 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 $t_{\text{timeslot}} \ll 1$. Therefore, we expect that $T_{\texttt{attempt}} \gg 1$ for most PGTs.

4.5.2. SERVICE MODEL

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

Definition 25 (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 §4.5.7.

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].

4.5.3. NETWORK CAPABILITY UPDATE

The evaluation will be performed on a network with a star topology (Figure 4.7), 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].

4.5.4. 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.

4.5.5. Demand Registration and Queuing

We use the following set of rules for demand registration:

Definition 26. *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.*

Definition 27. *Let the utilisation of a PGT τ with execution time E and period $T_{attempt}$ be $U_\tau = E/T_{attempt}$. 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.

4.5.6. 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 4.8.3) 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 $1 - \epsilon_{\text{service}}$. 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 4.8.3. Otherwise ($R > 0$), we simply set $R_{\texttt{attempt}} = p_{\texttt{packet}}^{-1} R$

4.5.7. Network Scheduling

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:

Definition 28 (Utilisation Bound)**.** *For all resources r, the utilisation of r, $\mathcal{U}_r$, must satisfy*

$$\mathcal{U}_r = \sum_{\tau : r \in \rho_\tau} U_\tau \leq \hat{\mathcal{U}} \tag{4.5}$$

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

Definition 29 (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 27 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 $\hat{\mathscr{U}}$ in 28 is restricted to $(0,1]$ as if any $\mathscr{U}_r > 1$, then there is not physically enough time to schedule all the required PGAs. The estimated computation time in 29 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 4.4.

We write $\mathscr{T}$ to be the set of PGTs for which PGAs must be scheduled in the next network schedule. Every PGT in $\mathscr{T}$ 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.

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 (4.4), 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 $\tau_i \in \mathscr{T}$, 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,

$$r_{i,j} = \sigma_i + \max\{(j-1)T_i, c_{i,j-1} + t_{\texttt{minsep,i}}\} \tag{4.6}$$

$$d_{i,j} = \sigma_i + jT_i \tag{4.7}$$

where $j = 1, \ldots, \lfloor (t_{\texttt{expiry},i} - \sigma_i)/T_i \rfloor$. 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 30 (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
```

4.5.8. 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].

4.6. 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 4.8.4. 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.

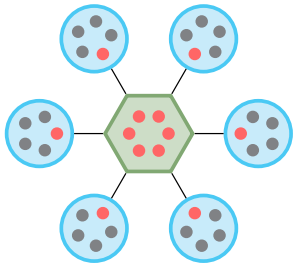


Figure 4.7: Example of a star-topology network with 6 nodes. The outer circles represent the end nodes and the central hexagon a central junction node. In each network time-slot the central node can attempt to create an entangled link with each of the end nodes, and then perform entanglement swaps to create end-to-end links between pairs of end nodes. The orange dots represent communication qubits and the black dots represent memory qubits.

4.6.1. METRICS

Definition 31 (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 32 (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.

Definition 33 (Metric: 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 $\mathscr{M} \subseteq \hat{\mathbb{S}}$ be the set of those application sessions which obtained minimal service. Then the metric, p^{MS}, is given by*

$$p^{MS} = \frac{|\mathscr{M}|}{|\hat{\mathbb{S}}|} \in [0,1]. \tag{4.8}$$

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.

Definition 34 (Metric: Average Time Spent in Queue)**.** *Let $t_{\texttt{submit},\mathscr{S}}$ be the time at which the demand corresponding to session $\mathscr{S}$ is submitted to the central controller. Let $t_{\texttt{exit},\mathscr{S}}$*

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

Table 4.2: 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}.

be the time at which the demand corresponding to session $\mathscr{S}$ leaves the queue. Then the metric is given by

$$\bar{t}_{queue} = \frac{1}{|\mathbb{S}|} \sum_{\mathscr{S} \in \mathbb{S}} \left(t_{\texttt{exit},\mathscr{S}} - t_{\texttt{submit},\mathscr{S}} \right). \tag{4.9}$$

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

4.6.2. Network Configuration

We consider a network consisting of 6 end nodes connected to a single central node as shown in Figure 4.7. 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 $p_{\text{gen}} \ll 1$.

4.6.3. Admission Control

Utilisation Bound

We set the value of the utilisation bound $\hat{\mathscr{U}} = 0.85$. A restriction to $\hat{\mathscr{U}} \leq 1$ 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

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

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 $\hat{\mathcal{U}} < 1$.

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 29. 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 4.8.4.

4.6.4. SESSION MODEL

CREATION TIME

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

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 4.3.2. The Qoala files we use for these test applications are given in Appendix 4.8.4. We fill out the remaining fields in the application sessions as follows:

$$\mathscr{S} = (\mathscr{N}, \texttt{APP}, N_{\texttt{inst}} = 100,\ t_{\texttt{expiry}} = t_{\texttt{submit}} + t_{\texttt{max duration}}) \tag{4.10}$$

where $\texttt{APP} \in \{\texttt{MDA}, \texttt{CKA}\}$, $t_{\texttt{submit}}$ 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.

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.

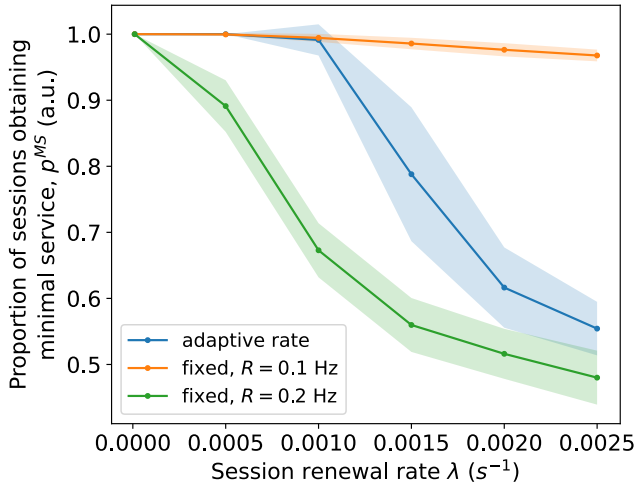
TERMINATION OF DEMANDS

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.

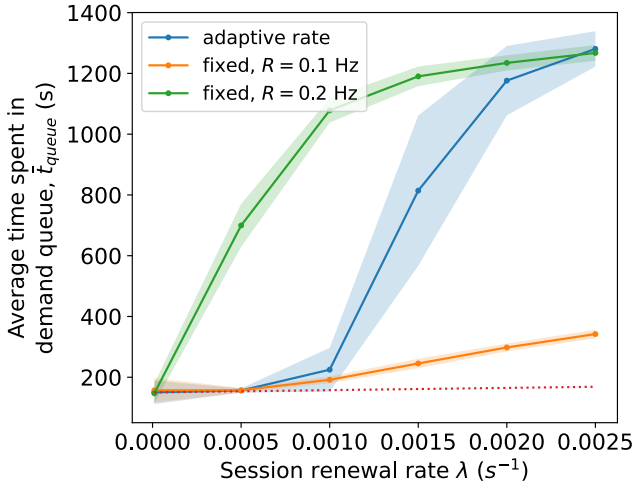
4.6.5. RESULTS

Simulation of the implementation (Section 4.5) 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 4.6.1. 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 4.8) to simulations where all nodes are running the CKA application in the client-server regime (Figure 4.9).

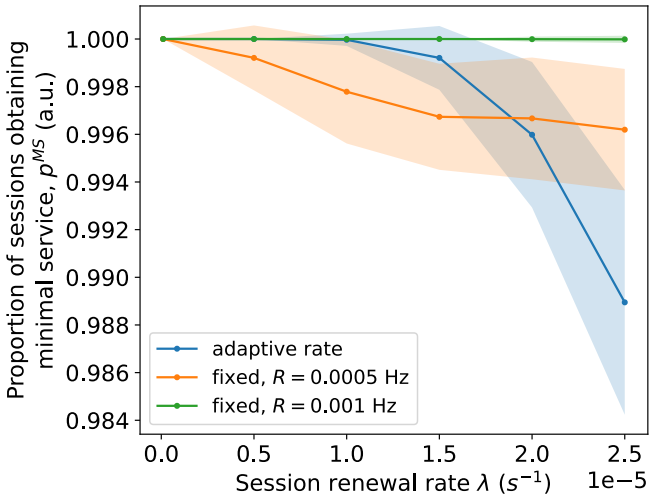


(a)

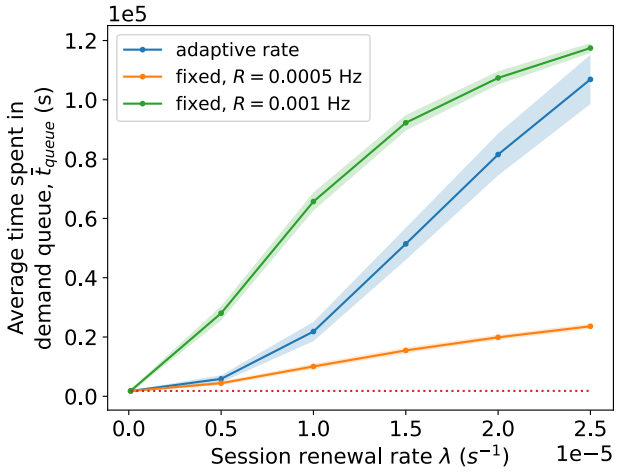


(b)

Figure 4.8: 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 $\bar{t}_{\text{queue}}$ 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.



(a)



(b)

Figure 4.9: 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 $\bar{t}_{\text{queue}}$ 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.

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 (4.8). 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 4.8a) 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 4.8a and 4.9a 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 4.9a, $p^{MS} = 1$ 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 4.8b and 4.9b confirm that the average time a demand spends in the queue, $\bar{t}_{\text{queue}}$ 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 4.8) 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 in lower p^{MS}. See §4.6.5 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 $\bar{t}_{\text{queue}}$. This is expected because a lower fixed rate again translates to lower load on the network. In section §4.6.5, we comment on the effectiveness of adaptive rate requests, in comparison to requests for fixed rates.

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 4.8b. 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.

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 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 $1 - \epsilon_{\text{service}}$. 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 4.8b and 4.9b, where a (sharp) increase in the value of $\bar{t}_{\text{queue}}$ occurs as λ passes the critical values above.

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 4.8, 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 schedule. 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 4.8a 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. [66, 93]).

4.7. 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 [66, 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.

4.8. APPENDICES

4.8.1. NOTATION

A summary of the notation used in the paper is given in Table 4.4.

4.8.2. WHO KNOWS WHAT

CENTRAL CONTROLLER

The central controller *knows* the following information:

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

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:

Symbol	*Definition*
	Application Sessions
$\mathcal{S}$	Application session
$\mathcal{N}$	Set of nodes in the network
`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
`MDA`	Measure Directly Application, based on QKD
`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 4.4: Notation used throughout this paper. Entries which fall into multiple categories are only shown once.

- Full network topology
- EGPs used on the network
- $p_{\texttt{packet}}$ (up to empirical deduction)

- Application sessions being run on other nodes.

4.8.3. METHODS FOR CREATING PACKET GENERATION TASKS

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 12. *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

$$Q'_L \approx Q'_2 \left(\frac{Q'_3}{Q'_2}\right)^{\frac{N}{m}-2} \tag{4.11}$$

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

$$\begin{aligned} Q'_2 = &\big(F_p(k-1;m,p)\big)^2 \\ &- (k-1)b(k;m,p)F_b(k-2;m,p) \\ &+ mpb(k;m,p)F_b(k-3;m-1,p) \end{aligned} \tag{4.12}$$

and

$$Q'_3 = \big(F_b(k-1,m,p)\big)^3 - A_1 + A_2 + A_3 - A_4 \tag{4.13}$$

where

$$A_1 = 2b(k;m,p)F_b(k-1;m,p)\Big\{(k-1)F_b(k-2;m,p)$$

$$- mpF_b(k-3;m-1,p)\Big\}$$

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

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

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

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.

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 $\epsilon_{\text{service}} \ll 1$, 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

$$\begin{aligned}\mathbb{P}[M'] &= \mathbb{P}[S_N < N_{\texttt{inst}}] \\ &= \mathbb{P}[\mathbb{E}[S_N] - S_N > \mathbb{E}[S_N] - N_{\texttt{inst}}] \\ &= \mathbb{P}[pN - S_N > N(p-\alpha)] \\ &\leq \exp\left(-2N(p-\alpha)^2\right) \end{aligned} \tag{4.14}$$

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

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

$$\min\left\{N \mid \epsilon_{\text{service}} > \exp\left(-2N(p-\alpha)^2\right) \wedge N > N_{\texttt{inst}}\right\} \tag{4.15}$$

from which we can then recover the minimum rate.

4.8.4. ADDITIONAL EVALUATION DETAILS

ENTANGLEMENT GENERATION MODEL

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

$$\rho = \frac{1-F}{3}\mathbb{I}_2 + \frac{4F-1}{3}|\psi\rangle\langle\psi|. \tag{4.16}$$

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 4.2.

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.

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 4.10.

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.

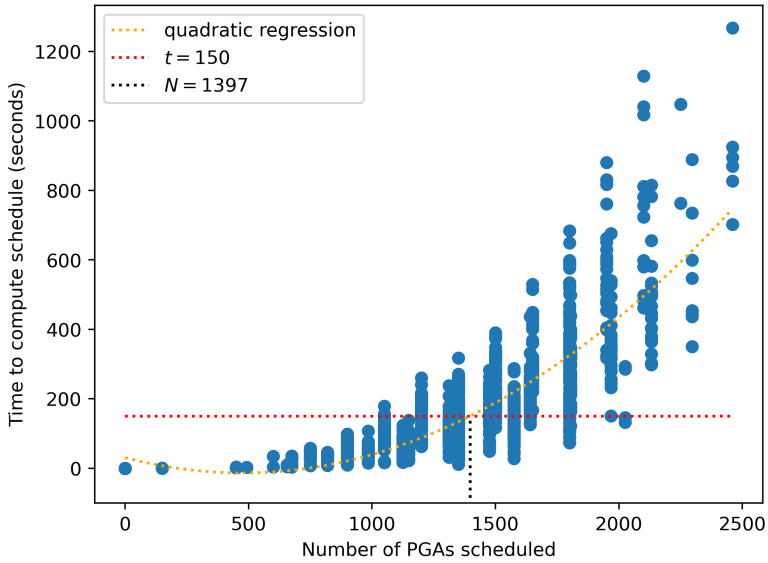


Figure 4.10: Times taken for our computational server to compute network schedules. The server has an Intel® Xeon® Gold 6230 CPU, with each core running at a maximum clock speed of 3.9GHz, and 189GB of random access memory. Each simulation was run on a single core, with up to 40 simulations being run in parallel. The regression modelling was performed using the `numpy` Python module

Choice of Expiry Times and Rates

Using the method in 4.8.3, we find that to have $\epsilon_{\text{service}} = 10^{-5}$, 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 $t_{\texttt{max duration}}$, $R^{\min}_{\epsilon_{\text{service}}}$.
  - approximately $2R^{\min}_{\epsilon_{\text{service}}}$

We choose to look at demands requesting a fixed packet generation rate of $R^{\min}_{\epsilon_{\text{service}}}$, as it serves as a good comparison to the adaptive rate (which would use this rate in the absence of queuing). The choice of $2R^{\min}_{\epsilon_{\text{service}}}$ 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, $R^{\min}_{\epsilon_{\text{service}}}$. 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 $\hat{\mathscr{U}}/5$, 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 28 or 29, 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 $\hat{\mathscr{U}}$ and the PGA cap exist.

4

SCHEDULER MODEL

We use the network scheduler as described in 4.5.7, 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.

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
  fidelity: 1.0
  typ: measure_directly
```

```
  role: create
```

Listing 4.1: Qoala file for Alice

```
META_START
    name: bob
    parameters: alice_id
    csockets: 0 -> alice
    epr_sockets: 0 -> alice
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: {alice_id}
  epr_socket_id: 0
  num_pairs: 1
  virt_ids: all 0
  timeout: 1000
  fidelity: 1.0
  typ: measure_directly
  role: receive
```

Listing 4.2: 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 4.3: 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 4.4: Qoala file for Server

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 Binomial($N, p_{\texttt{packet}}$) 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.

CALCULATING THE EXPECTED QUEUING TIME

We use the notation from § 4.6.4. Let $t_{\text{renew}} \sim \text{Exponential}(\lambda)$. Then the time which a demand $\mathscr{D}$ would sit in the demand queue for in the absence of other demands is $Q \equiv T_{SI} - t_{\text{renew}} \mod T_{SI}$. For convenience we write $T_{SI} = T$, and let $t_{\mathscr{D}}$ be the time demand $\mathscr{D}$ is submitted. We then calculate:

$$\begin{aligned} F_Q(\tau) &= \mathbb{P}[Q \leq \tau] \\ &= \mathbb{P}\left[t_{\mathscr{D}} \in \bigcup_{k=1}^{\infty} [kT - \tau, kT] \right] \\ &= \sum_{k=1}^{\infty} \int_{kT-\tau}^{kT} \lambda e^{-\lambda x} \mathrm{d}x \\ &= \sum_{k=1}^{\infty} e^{-kT\lambda}(-1 + e^{\lambda\tau}) \\ &= \frac{-1 + e^{\lambda\tau}}{-1 + e^{\lambda T}} \end{aligned} \tag{4.17}$$

$$\begin{aligned} \mathbb{E}[Q] &= \int_0^T \tau \, \mathrm{d}F_Q \\ &= \int_0^T \frac{\tau\lambda e^{\lambda\tau}}{-1 + e^{\lambda T}} \, \mathrm{d}\tau \\ &= T\left(1 - \frac{1}{\lambda T} + \frac{1}{e^{\lambda T} - 1}\right) \end{aligned} \tag{4.18}$$

REFERENCES

[1] A. K. Ekert, *Quantum cryptography based on Bell's theorem,* Physical Review Letters **67**, 661 (1991).

[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 **04**, 883 (2006).

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

[5] A. M. Childs, *Secure assisted quantum computation,* Quantum Information and Computation **5** (2005), 10.26421/QIC5.6.

[6] S. Tani, H. Kobayashi, and K. Matsumoto, *Exact Quantum Algorithms for the Leader Election Problem,* ACM Transactions on Computation Theory **4**, 1 (2012).

[7] S. Wehner, D. Elkouss, and R. Hanson, *Quantum internet: A vision for the road ahead,* Science (2018), 10.1126/science.aam9288.

[8] A. Einstein, B. Podolsky, and N. Rosen, *Can quantum-mechanical description of physical reality be considered complete?* Physical Review **47**, 777 (1935).

[9] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition* (Cambridge University Press, 2010).

[10] W. Dür, H.-J. Briegel, J. I. Cirac, and P. Zoller, *Quantum repeaters based on entanglement purification,* Physical Review A **59**, 169 (1999).

[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 **372**, 259 (2021).

[12] Á. G. Iñesta, G. Vardoyan, L. Scavuzzo, and S. Wehner, *Optimal entanglement distribution policies in homogeneous repeater chains with cutoffs,* npj Quantum Information **9**, 1 (2023).

[13] C. Cabrillo, J. I. Cirac, P. García-Fernández, and P. Zoller, *Creation of entangled states of distant atoms by interference,* Physical Review A **59**, 1025 (1999).

[14] C. E. Bradley, S. W. de Bone, P. F. W. Möller, 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,* npj Quantum Information **8**, 122 (2022).

[15] C. Delle Donne, M. Iuliano, B. van der Vecht, G. M. Ferreira, H. Jirovská, T. J. W. van der Steenhoven, A. Dahlberg, M. Skrzypczyk, D. Fioretto, M. Teller, P. Filippov, A. R.-P. Montblanch, J. Fischer, H. B. van Ommen, N. Demetriou, D. Leichtle, L. Music, H. Ollivier, I. te Raa, W. Kozlowski, T. H. Taminiau, P. Pawełczak, T. E. Northup, R. Hanson, and S. Wehner, *An operating system for executing applications on quantum network nodes,* Nature **639**, 321 (2025).

[16] B. van der Vecht, A. T. Yücel, H. Jirovská, and S. Wehner, *Qoala: an application execution environment for quantum internet nodes,* (2025), 2502.17296 [quant-ph] .

[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).

[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).

[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)*, 2023 32nd International Conference on Computer Communications and Networks (ICCCN) , 1 (2023).

[20] A. Pirker and W. Dür, *A quantum network stack and protocols for reliable entanglement-based networks,* New Journal of Physics **21**, 033003 (2019).

[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 **110** (2024), 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 **8**, 1 (2022).

[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* (2019) pp. 159–173.

[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), 10.1109/TQE.2024.3358674.

[26] J. I. Naus, *Approximations for Distributions of Scan Statistics,* Journal of the American Statistical Association **77**, 177 (1982).

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

[28] M. Skrzypczyk and S. Wehner, *An Architecture for Meeting Quality-of-Service Requirements in Multi-User Quantum Networks,* (2021).

[29] C. Cicconetti, M. Conti, and A. Passarella, *Request Scheduling in Quantum Networks,* IEEE Transactions on Quantum Engineering **2**, 2 (2021).

[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)* (2022) pp. 341–352.

[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 **103**, 14 (2015).

[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 **38**, 69 (2008).

[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*, Request for Comments RFC 5810 (Internet Engineering Task Force, 2010).

[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.

[35] Z. Yang and C. Cui, *Reconfigurable Quantum Internet Service Provider,* in *ICC 2023 - IEEE International Conference on Communications* (2023) pp. 3497–3503.

[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 **3**, 1 (2022).

[37] I. Monga, E. Saglamyurek, E. Kissel, H. Haffner, and W. Wu, *QUANT-NET: A testbed for quantum networking research over deployed fiber,* in *Proceedings of the 1st Workshop on Quantum Networks and Distributed Quantum Computing* (ACM, New York NY USA, 2023) pp. 31–37.

[38] R. S. Tessinari, R. I. Woodward, and A. J. Shields, *Software-defined quantum network using a QKD-secured SDN controller and encrypted messages,* (2023).

[39] V. Martin, J. P. Brito, L. Ortíz, R. B. Méndez, J. S. Buruaga, R. J. Vicente, A. Sebastián-Lombraña, D. Rincón, F. Pérez, C. Sánchez, 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. Abellán, M. Hentschel, J. M. Rivas-Moscoso, A. Pastor-Perales, J. Folgueira, and D. López, *MadQCI: a heterogeneous and scalable SDN-QKD network deployed in production facilities,* npj Quantum Information **10**, 80 (2024).

[40] 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).

[41] 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.

[42] 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 **35**, 1357 (2017).

[43] H. Wang, Y. Zhao, and A. Nag, *Quantum-key-distribution (QKD) networks enabled by software-defined networks (SDN),* Applied Sciences **9**, 2081 (2019).

[44] 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.

[45] H. S. Hadi and A. J. Obaid, *Quantum key distribution (QKD) for wireless networks with software-defined networking,* Internet Technology Letters **n/a**, e547 (2024).

[46] 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.

[47] 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 **14**, 16752 (2024).

[48] Quantum Key Distribution Industry Specification Group, *Quantum Key Distribution (QKD): Control Interface for Software Defined Networks*, Tech. Rep. (European Telecommunications Standards Institute, 2024).

[49] S. S. Craciunas and R. S. Oliver, *Combined task- and network-level scheduling for distributed time-triggered systems,* Real-Time Systems **52**, 161 (2016).

[50] H. J. Kramer, *TTEthernet (time-triggered ethernet) - eoPortal,* (2022).

[51] TTTech, *NASA's orion spacecraft,* .

[52] M. Ruf, N. H. Wan, H. Choi, D. Englund, and R. Hanson, *Quantum networks based on color centers in diamond,* Journal of Applied Physics **130**, 070901 (2021).

[53] 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 **372**, 259 (2021).

[54] 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. **130**, 050803 (2023).

[55] 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 **3**, 538 (2007).

[56] J. P. Covey, H. Weinfurter, and H. Bernien, *Quantum networks with neutral atom processing nodes,* npj Quantum Information **9**, 1 (2023).

[57] 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 **122** (2016), 10.1007/s00340-015-6299-2.

[58] 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 **438**, 828 (2005).

[59] 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 **316**, 1316 (2007).

[60] 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 **132** (2024), 10.1103/physrevlett.132.150604.

[61] 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 **5**, 186 (2022).

[62] G. Vardoyan, S. Guha, P. Nain, and D. Towsley, *On the Stochastic Analysis of a Quantum Entanglement Distribution Switch,* IEEE Transactions on Quantum Engineering **2**, 1 (2021).

[63] G. Vardoyan, S. Guha, P. Nain, and D. Towsley, *On the exact analysis of an idealized quantum switch,* Performance Evaluation **144**, 102141 (2020).

[64] 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.

[65] 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 **8** (2022), 10.1038/s41534-022-00582-8.

[66] S. Gauthier, G. Vardoyan, and S. Wehner, *An architecture for control of entanglement generation switches in quantum networks,* IEEE Transactions on Quantum Engineering **4**, 1 (2023).

[67] S. Gauthier, T. Vasantam, and G. Vardoyan, *On-Demand Resource Allocation for a Quantum Network Hub,* IEEE Transactions on Quantum Engineering **7**, 1 (2026).

[68] L.-M. Duan, M. D. Lukin, J. I. Cirac, and P. Zoller, *Long-distance quantum communication with atomic ensembles and linear optics,* Nature **414**, 413 (2001).

[69] 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. **98**, 190503 (2007).

[70] N. Sangouard, C. Simon, H. de Riedmatten, and N. Gisin, *Quantum repeaters based on atomic ensembles and linear optics,* Rev. Mod. Phys. **83**, 33 (2011).

[71] N. Sangouard, R. Dubessy, and C. Simon, *Quantum repeaters based on single trapped ions,* Phys. Rev. A **79**, 042340 (2009).

[72] 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,* Physical Review Research **5**, 023086 (2023).

[73] H. Zimmermann, *Osi reference model - the iso model of architecture for open systems interconnection,* IEEE Transactions on Communications **28**, 425 (1980).

[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 **497**, 86 (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 **558**, 268 (2018).

[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 **607**, 69 (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*, OSDI '06 (USENIX Association, USA, 2006) p. 89–102.

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

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

[80] S. Shenker, *Fundamental design issues for the future Internet,* IEEE Journal on Selected Areas in Communications **13**, 1176 (1995).

[81] A. Furusawa, J. L. Sørensen, S. L. Braunstein, C. A. Fuchs, H. J. Kimble, and E. S. Polzik, *Unconditional quantum teleportation,* Science **282**, 706 (1998), https://www.science.org/doi/pdf/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 **429**, 737 (2004).

[83] D. Leichtle, L. Music, E. Kashefi, and H. Ollivier, *Verifying BQP Computations on Noisy Devices with Minimal Overhead,* PRX Quantum **2**, 040302 (2021).

[84] F. B. Talbot, *Resource-Constrained Project Scheduling with Time-Resource Tradeoffs: The Nonpreemptive Case,* Management Science **28**, 1197 (1982).

[85] H. Jirovská, *Evaluating an RCPSP Implementation of Quantum Program Scheduling,* Master's thesis, Technische Universiteit Delft, Delft (2023).

[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 **23**, 9818 (2023).

[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 **12**, 12731 (2022).

[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 , 1 (2024).

[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 **5**, 25 (2019).

[90] R. Van Meter, T. Satoh, T. D. Ladd, W. J. Munro, and K. Nemoto, *Path Selection for Quantum Repeater Networks,* Networking Science **3**, 82 (2013).

[91] J. A. Stankovic, M. Spuri, G. C. Buttazzo, and K. Ramamritham, *Deadline scheduling for real-time systems: EDF and related algorithms*, The Kluwer international series in engineering and computer science No. SECS 460 (Kluwer Academic Publishers, New York, 1999).

[92] C. L. Liu and J. W. Layland, *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment,* Journal of the ACM **20**, 46 (1973).

[93] S. Low and D. Lapsley, *Optimization flow control. i. basic algorithm and convergence,* IEEE/ACM Transactions on Networking **7**, 861 (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 Signaling and Management of ATM Networks, **29**, 539 (1997).

[95] M. Christandl and S. Wehner, *Quantum Anonymous Transmissions,* in *Advances in Cryptology - ASIACRYPT 2005*, Vol. 3788, edited by 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 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2005) pp. 217–235.

[96] V. Lipinska, G. Murta, and S. Wehner, *Anonymous transmission in a noisy quantum network using the W state,* Physical Review A **98**, 052320 (2018).

[97] W. Hoeffding, *Probability inequalities for sums of bounded random variables,* Journal of the American Statistical Association **58**, 13 (1963).

[98] R. F. Werner, *Quantum states with Einstein-Podolsky-Rosen correlations admitting a hidden variable model,* Physical Review A **40**, 4277 (1989).

5

ARQON: A SUITE OF CONTROL APPLICATIONS ENABLING A RELIABLE QUANTUM NETWORK

A quantum network's purpose is to enable users to execute applications on end nodes. This requires the network to provide the service of creating entangled links between those nodes. Users of mature networks, such as the internet or the telephone network expect accepted service demands to be met reliably. We first define reliability requirements that extend classical computer network concepts to quantum network service delivery. We then introduce Arqon, a suite of control applications designed to deliver reliable service in centrally controlled quantum networks. We demonstrate through both analytic and numerical evaluation that Arqon satisfies all reliability requirements for accepted demands. These evaluations consider static network topologies. We provide a complete Python implementation and perform complexity analysis showing that admission control scales as $O(k^3)$ in the number of incoming demands k and schedule computation scales as $O(N^3)$ in the number of accepted demands to schedule N.

5.1. INTRODUCTION

A quantum network enables users to execute applications that enable new functionalities. These include secure remote computation [1–3], secure communication which does not rely on computational assumptions [4, 5], fast coordination of decisions between remote parties without the real-time exchange of messages [6–10], anonymous leader election [11], and improvements in the precision of metrology [12, 13]. These applications require quantum communication between quantum devices, supplementing the classical communication which underpins classical network applications. Entangled links between end nodes, known as end-to-end links, are key resources that can enable arbitrary quantum communication operations. The service that a quantum network must provide to end nodes is to create these end-to-end entangled links according to the quality, quantity, and

time based requirements of the end node applications.

To correctly schedule the quantum communication operations needed in local application execution, the operating systems of programmable end nodes may rely on a network schedule. This schedule dictates when the nodes can access internal resources of the network to attempt end-to-end entangled link generation. In [14], Beauchamp *et al.* proposed a centrally controlled quantum network architecture which is designed to provide this service in a way that is compatible with the state of the art operating systems for quantum processing nodes, QNodeOS [15] and its upgrade Qoala [16].

The network architecture in [14] is modular, in the sense that it is defined to consist of a set of core functionalities, which may be implemented in separate control applications that can each be modified or upgraded independently of the others. A control application packages related network functionalities such as admission control and schedule computation, and provides specific algorithms implementing these functionalities. The network architecture is centered around the periodic computation and distribution of network schedules, enabling internal resources and end nodes to receive schedules in advance of the times at which they can be executed. In a proof of principle implementation and evaluation [14], the proportion of satisfied demands decreased substantially as the number of submitted demands increased, exposing a need for improved control applications. More broadly, an outstanding challenge in the design of quantum network architectures is to develop control applications that can deliver reliable service to end nodes.

In computer networking, a system or protocol is reliable if it is able to consistently deliver services to their intended recipients without error or significant delay [17]. Adapting this principle to the service of a quantum network, we define the essential features of reliable service:

(R1) demands receive accept/reject responses;

(R2) accepted demands are satisfied with a tolerably high probability;

(R3) accepted demands are satisfied before their deadlines;

(R4) and the amount or frequency of demands from other users does not disrupt service to accepted demands.

Arqon is a suite of control applications for the centrally controlled quantum network architecture in [14], which is designed to overcome the challenge of delivering reliable service to end nodes. Arqon comprises four control applications: A *Network Manager*, a *Demand Manager*, a *Network Scheduler*, and a *Schedule Manager*. The Network Manager is responsible for maintaining an accurate overview of the network, including all components and the entanglement generation capabilities of the components. The Demand Manager is responsible for all processes that explicitly relate to demands, such as registration of new demands, creating accept/reject responses to send to end nodes, and re-formatting demands into an internal representation. The Network Scheduler produces network schedules based on end node demands. In addition to a *Compute Schedule* process, the Network Scheduler also includes an *Admit Tasks* process that determines whether demands are accepted for service or rejected. Finally, the Schedule Manager directs the distribution of network schedules to all network components.

In this work we make the following contributions:

- We introduce *Arqon*, the first suite of control applications for a quantum network architecture capable of delivering reliable service to all accepted demands. To make this achievement precise, we propose a set of reliability requirements for service from a quantum network. We deliver a full performance analysis of our Network Scheduler control application, proving that it can meet the proposed requirements. Moreover, we validate in simulation Arqon's performance for a variety of network topologies and sets of demands.

- We provide a Python implementation of Arqon that enables researchers to simulate realistic multi-user quantum networks, investigate performance under diverse traffic patterns, and develop extensions to our control algorithms. In particular, the Arqon simulator allows configuration of a network topology, the entanglement generation capabilities (success rate and fidelity) between end nodes, and sets of end node application requirements.

- We prove that the operational complexity of our implementation of the Network Scheduler is
$$O\Big(k2^R\big((N+k)^2+R\big)+(N+R)^2+N^3R\Big),$$
where N is the current number of accepted demands to schedule, k is the number of internal representations of new demands requiring an accept/reject decision, and R is the number of internal resources in the network (network components that are not end nodes). In our evaluations, we observe that the time to produce a network schedule (t_{compute}) is always significantly less than the allowed time, known as a scheduling interval (T^{SI}). That is, $t_{\text{compute}} \ll T^{SI}$.

The notation used throughout this paper is summarized in Table 5.4, included in Appendix 4.8.1. The rest of this paper is organized as follows: In Section 5.2 we discuss related work. In Section 5.3 we discuss design considerations for Arqon. In Section 5.4 we define each control application in turn. In Section 5.5 we state and prove performance results for the abstractly defined control applications. In Section 5.6 we discuss a specific implementation of Arqon in Python. In Section 5.7 we provide an operational complexity analysis of this implementation. In Section 5.8 we discuss results from numeric simulations. Finally, we conclude in 5.9 and discuss our outlook on future research goals.

5.2. RELATED WORK

The Arqon suite of control applications is designed to integrate with the centrally controlled quantum network architecture defined in [14]. This architecture is compatible with the network protocol stack proposed by Dahlberg *et al.* in [18], where the stack is decomposed into layers, each of which exposes a functionality to the layer above and exploits the functionality of the layer below, as in the classical OSI network model [19]. We make use of the terminology of this network stack. The *physical layer* of the network is responsible for attempting entanglement generation. Above the physical layer is the *link layer*, which manages link level entanglement generation, transforming it into a robust service. The link layer service was experimentally realized in Pompili *et al.* [20],

demonstrating it's viability as a functional layer of a quantum network. Operations such as scheduling entanglement swaps in a repeater chain and aligning entanglement generation with the memory management service of a quantum node are functions of the link layer. The network architecture [14] defines a centrally controlled structure for the *network layer*, which is responsible for coordinating end-to-end entanglement generation. The combination of this architecture with the Arqon suite of control applications constitutes an implementation of the network layer.

Experiments on leading quantum network hardware platforms present a major technological challenge: the expected time to generate an entangled link between neighboring nodes at metropolitan distance scales is almost as long as the maximum storage time. In a state-of-the-art demonstration based on trapped ion nodes [21] the average reported time to generate one entangled pair over an effective 10 km link was 450 ms, while the average maximum storage time was 550 ms with a standard deviation of 36 ms. In networks where the maximum storage time of entangled links does not exceed by orders of magnitude the time required to generate such a link, the multiple neighboring entangled links required to construct an end-to-end link will rarely co-exist. To overcome this technical challenge, Arqon produces network schedules that simultaneously allocate nodes along end-to-end routes for durations of time calculated based on end-to-end entanglement generation rates. This method of resource allocation is reminiscent of circuit switching in wired telephone networks [22, 23], Frame Relay [24, 25], and Autonomous Transfer Mode (ATM) networks [23, 26, 27].

In wired telephone networks, multiple calls share physical links through time division multiplexing (TDM), where each call receives a guaranteed time slot within a frame that repeats at fixed frequency for the entire call duration [22, 23]. The maximum number of simultaneous calls is determined by the number of time slots per frame on each link along a route. In ATM networks, multiple virtual circuits share physical links through statistical multiplexing of 53-byte cells transmitted on demand rather than on fixed schedules [23, 26]. Users requesting virtual circuits provide traffic contracts specifying bounds on data production: sustained cell rate (average long-term rate), peak cell rate (maximum rate), and maximum burst size (longest burst at peak rate). Admission control evaluates these contracts and, if accepted, calculates effective bandwidth to reserve along the entire route for the virtual circuit's duration [27]. Frame Relay uses similar mechanisms to ATM but with variable-length frames (1 to 8,192 bytes) and a simpler single-service-class quality of service model with three parameters: a guaranteed data transfer rate (Committed Information Rate (CIR)), an allowed burst size at the CIR and a maximum amount that a burst may exceed the CIR [24, 25]. In contrast, Arqon schedules exclusive end-to-end physical access for demand-dependent durations separated by demand-dependent minimum separation intervals when routes fully release access. Network-wide schedules are pre-computed sequentially for extended scheduling intervals that are long relative to typical access durations. Arqon's admission control evaluates user demands, with accepted demands receiving service agreements guaranteeing a minimum scheduling frequency per interval while respecting minimum separation requirements.

Many proposals of quantum network architectures and protocols, including those of [28–35], have considered the problem of coordinating the generation of end-to-end entangled links, requiring the use of network resources. Van meter *et al.* in [33] further

proposed a framework for routing of entanglement across many smaller networks, and identified the types of protocols required to do so. A common feature of these architectures and protocols is that they do not aim to provide end nodes with a guaranteed quality of service and they do not develop protocols for preventing arriving requests from disrupting service to existing requests. In contrast, Arqon is designed to provide end nodes with guaranteed quality of service agreements and incorporates an admission control process that can reject arriving demands to prevent disrupting existing service agreements.

In [36] Skrzypczyk *et al.* defined a quantum network architecture for meeting quality of service requirements in multi-user quantum networks. Quality of service was defined with respect to classically inspired metrics (throughput and jitter of entangled link generation) and quantum specific metrics (fidelity of generated entanglement). The architecture employs Time-Division Multiple-Access (TDMA) [37] scheduling with fixed-duration time slots, granting contention-free access to all qubits along a demand's assigned route for integer numbers of consecutive slots. The authors identified the need for an admission control protocol to prevent arriving demands from disrupting service to existing demands. However, no admission control protocol was defined or implemented, leaving the architecture unable to satisfy reliability requirement *(R4)*. In contrast, Arqon defines a concrete admission control protocol which rejects demands that cannot be satisfied without disrupting service to accepted demands.

5.3. DESIGN CONSIDERATIONS

Our task is to design a suite of control applications, which when integrated into the modular architecture defined in [14], create a quantum network satisfying all of the reliability requirements *(R1)*-*(R4)*. From these requirements we derive the following core considerations which guide the design of the Arqon suite of control applications.

5.3.1. CORE CONSIDERATIONS

(C1) Algorithms that handle time-sensitive network scheduling must execute within bounded time, requiring operational complexity that is polynomial in the number of demands.

(C2) Service agreements must be created for accepted demands, defining the type and quality of service that Arqon commits to provide. In complement, rejection messages must be sent to rejected demands.

(C3) A demand may be accepted by Arqon if and only if acceptance will not disrupt any existing service agreements.

(C4) Arqon must support dynamic updates of the network topology by allowing new nodes to register and removing network components that fail or become unresponsive.

Service agreements will be formally defined in Section 5.4, following necessary preliminaries.

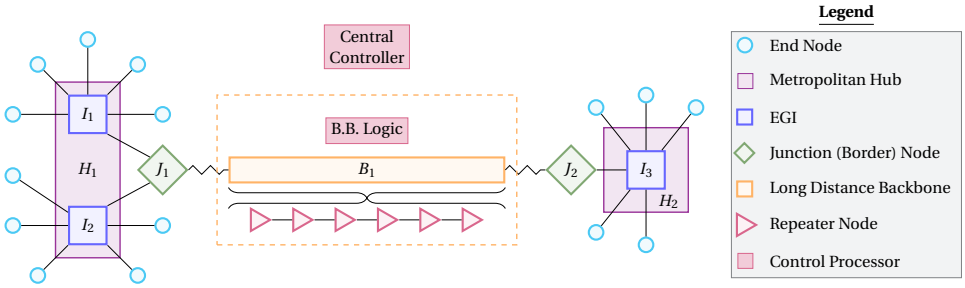


Figure 5.1: Example quantum network resource graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ for a quantum network where the resources to be shared are a long distance backbone channel B_1, two junction nodes J_1, J_2, and three entanglement generation interfaces I_1, I_2, I_3 grouped into two metropolitan hubs (H_1 and H_2). Edges represent logical connections, that is if $(v_1, v_2) \in \mathcal{E}$, then it is possible to create an optical path between components v_1 and v_2, possibly via some optical switch.

5

5.3.2. Network Model

We consider a centrally controlled quantum network where user controlled end nodes submit demands encoding application execution requirements [14]. Demands are not for single entangled pairs, but rather for repeated creation of packets containing multiple entangled links, with a minimum separation time between packets, until the demand expires. The network can be classified as a *generate-when-requested* network, where end-to-end entangled links are created to serve these demands. This means 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. In the following sub-section, we introduce the network components that can be used to construct a general purpose quantum network.

Network Components

At the physical layer, a general quantum network may consist of a heterogeneous mixture of quantum nodes and the interfaces between them, in an arbitrary topology. Figure 5.1 represents a possible network topology as a graph, where network components are illustrated as colored nodes and edges connecting components indicate logical connections. We consider the following network components:

- **End Nodes** These devices execute programs realizing part of a quantum network application, operate under independent (local) control, and accept input from users. An end node may be a *quantum processing node* with some quantum memory capabilities, as in [38–45], or a device without quantum memory which is capable of preparing/measuring photons, as in [46, 47]. Any end node can also perform classical operations, such as arithmetic operations and classical communication. Resources within the end nodes are managed locally, for example by the operating system QNodeOS [15] or its upgrade Qoala [16], and are therefore not under the direct control of any network controller. The local control includes a quantum network agent, which is a process responsible for communicating with a central

network controller. This agent submits demands to the central network controller, retrieves and installs network schedules and processes network status messages.

- **Central Controller** The purpose of the central controller is to produce network schedules in response to demands from end nodes. To do so, the central controller also maintains information about the overall network topology and the entanglement generation capabilities of each component in the network. The central controller hosts a suite of control applications which implement its various functionalities.

- **Metropolitan Hubs** A metropolitan hub facilitates a scalable method of multi-user network access [48, 49]. A metropolitan hub contains a logical control element which enforces the network schedule by controlling the physical devices used to realize the schedule. These devices are:

  - **Entanglement Generation Interfaces (EGIs)** These devices mediate entangled link generation between neighboring nodes via remote entanglement generation protocols [50, 51]. Different protocols require different EGIs: detection-in-midpoint protocols (e.g., heralded entanglement generation [52, 53]) use Bell-State Analyzers (BSAs) [54–56]; source-in-midpoint protocols [51] use entangled photon sources; and sender-receiver protocols use only optical channels [50, 57, 58]. Each EGI can only facilitate the generation of entangled links between a single pair of nodes at any one time.

  - **Switches** A switch is a reconfigurable interface that routes signals between different paths. Optical switches are compatible with fiber-based or free-space implementations (where light travels through air or vacuum). In both cases, an optical switch enables dynamic, high-speed reconfiguration of signal paths, supporting efficient routing. Switches allow the metropolitan hub to allocate pairs of nodes optical access to EGIs according to the network schedule.

- **Long-Distance Backbones** A long-distance backbone is a repeater chain consisting of a linear sequence of *repeater nodes*, such as those in [52, 53, 59–61]. The network treats each backbone as a single component that produces entangled links between its two border nodes [62] at a fixed rate and average fidelity. A long-distance backbone controller configures the repeater chain according to a repeater policy (see e.g. [63–65]). The repeater nodes and border nodes are automated components that strictly implement a repeater policy without supporting additional control logic.

- **Junction Nodes** A junction node is a border node equipped with additional control logic to support network branching. They provide an interface between multiple repeater chains and between repeater chains and a metropolitan hub. They have the physical capabilities to execute remote entanglement generation protocols, store the resulting entangled links, and perform entanglement swaps. The logical controller of a junction node manages local resources (such as quantum memories) to execute the network schedule.

INDEPENDENTLY PROGRAMMED END NODES

Multi-node quantum network applications are partitioned into separate single-node programs that are run concurrently on different end nodes. The central controller is not aware of the programs on independent end nodes. Each program contains instructions for local classical and quantum computations, and programs on separate end nodes interact with each other only through classical messaging and entanglement generation [16].

This hybrid classical-quantum programming model has an important implication for resource management: programs can be structured into sub-routines such that quantum states required by one sub-routine need not persist through following sub-routines. Individual quantum states therefore need not be stored for the entire program duration.

5.3.3. PHYSICAL CONSIDERATIONS

Generation of end-to-end entanglement requires implementing some intermediary protocols, in particular performing many physical attempts to generate entanglement between neighboring network components. Each of these physical attempts only succeeds probabilistically, resulting in some average rate of successfully generating end-to-end entangled links. Successfully generated entangled links are stored in quantum memories, which are subject to time-dependent errors due to environmental interactions [66, 67]. These decoherence processes limit the storage time, characterized by the memory coherence time [68–70]. To execute a sub-routine of an application program that requires a packet of entangled links, all links must remain sufficiently coherent throughout the duration of the sub-routine. Consequently, all links in a packet must be generated within a time window w determined by both the quantum memory coherence times and the sub-routine execution duration. The probability that a batch of attempts to generate end-to-end entangled links succeeds and the packet generation window w are not simply modeling parameters, they are inherited from the physical capabilities of the network components.

5.3.4. TIMESCALES

The time required for network elements to complete actions can only be estimated with finite precision. At the physical layer, actions have precisely characterized durations, allowing for accurate synchronization between multiple nodes, with timing precision ranging from tens of picoseconds (ps) to microseconds (μs) to milliseconds (ms), depending on the operations [15, 39]. Precise timing of a sequence of operations is crucial for processes like entanglement generation [40, 41, 44, 45, 71–73].

In contrast, at higher layers of the network stack, actions have variable durations and latencies, limiting feasible timing precision to μs or ms [74–76]. In particular, the time required to send a message over the internet varies, and a difference of several ms may occur in the communication times of repeated messages between a fixed sender and receiver.

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. The large differences in timescales between the physical layer and higher layers of the network stack imply that scheduling and triggering of actions at the physical layer is not a function of any

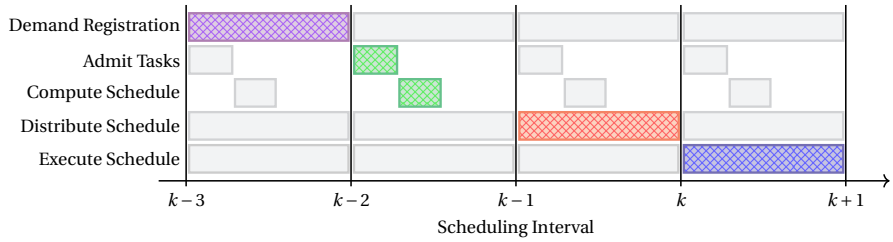


Figure 5.2: Process sequence resulting in execution of a network schedule in scheduling interval k. The hatched, colorful processes are executed in the indicated scheduling intervals in order to produce and distribute the schedule which will be executed in scheduling interval k. The grey processes highlight the periodic nature of our architecture.

higher layer of the network stack, but is the responsibility of dedicated and sophisticated electronics that are key components of the physical layer.

5.3.5. ARCHITECTURE

We state key features of our network architecture [14], as we make frequent reference to these features when defining the Arqon control 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.

APPLICATION SESSIONS

The aim of users of a quantum network is to successfully run applications. To do so, they execute hybrid classical-quantum programs on independent end nodes. As many hybrid programs have probabilistic outcomes, it is often required to execute the same program many times to extract a useful and reliable output. Realizing a single execution of a quantum network application requires each collaborating end node to successfully execute their local program. We refer to each of these individual executions of an application as an *application instance*. In practice, applications are typically associated with a deadline, before which all application instances must be executed. To capture these requirements, we define an *application session*.

Definition 35 (Application Session)**.** *Suppose end nodes* $\mathcal{N} = (\texttt{node1}, \texttt{node2}, ...)$ *wish to execute application* `App` *at least* $N^{\texttt{inst}}$ *times, before time* $t^{\texttt{expiry}}$*.*

Then we write the corresponding **application session** *as*

$$\mathcal{S} = (\texttt{session_id}, \mathcal{N}, \texttt{APP}, N^{\texttt{inst}}, t^{\texttt{expiry}}), \tag{5.1}$$

where `session_id` *is a unique identifier for the session.*

To ensure feasibility, it is necessary that the specified expiry time, $t^{\texttt{expiry}}$, is longer than any time-scale of the network. To establish a common understanding of how to locally configure programs for executing an application session (5.1) and ensure appropriate resource allocation, the end nodes in $\mathcal{N}$ execute two processes: a *network capability update* and *capability negotiation* [14]. In the network capability update, each node

independently queries a network capability table obtained from the central controller to retrieve end-to-end entanglement generation capabilities for each suitable path, including the average entanglement generation rate and the corresponding minimum fidelity of entangled pairs. In capability negotiation, the end nodes in $\mathcal{N}$ communicate to coordinate their local resource requirements for executing application programs, using the acquired network capabilities as input.

APPLICATION DEMANDS

Based on their requirements, application sessions on end nodes submit demands for packets of entanglement generation to the network. A distinct group of end nodes may simultaneously host multiple application sessions, hence the group may be associated with multiple demands at once.

Definition 36 (Demand)**.** *Application sessions register* **demands**. *These demands have the format*

$$d^{\text{raw}} = (\mathfrak{p}; t^{\texttt{minsep}}; t^{\texttt{expiry}}; N^{\texttt{inst}}) \tag{5.2}$$

where $\mathfrak{p} = (w, s, F)$ *defines a packet of* s *pairs to be generated within time window* w*, with minimum fidelity* F*;* $t^{\texttt{minsep}}$ *is a minimum time separation between attempts to generate packets,* $t^{\texttt{expiry}}$ *is the expiry time of the application session,* $N^{\texttt{inst}}$ *is the number of successes required to satisfy the demand. A demand is associated with implementation specific metadata, denoted* M_d*. The full demand can then be represented as*

$$d = (d^{\text{raw}}, M_d). \tag{5.3}$$

The metadata accompanying a demand is implementation specific and should include source and destination identifiers for the end nodes. Analogously to the metadata which accompanies a data packet in the classical internet, metadata can be used to inform control processes, including routing.

In our original formulation [14], the raw demand included a parameter $R_{\texttt{packet}}$ specifying the average requested rate of successful packet generation until the expiry time $t^{\texttt{expiry}}$. We have removed $R_{\texttt{packet}}$ from (5.2), as the number of instances $N^{\texttt{inst}}$ and expiry time $t^{\texttt{expiry}}$ together specify an average rate of packet generation.

As a new addition to the model from [14], we introduce a service error parameter, $\epsilon_d^{\text{service}}$, associated with a demand d. This parameter allows precise specification of the reliability requirement *(R2)* for each demand d,

$$\mathbb{P}[\text{demand d is satisfied}] \geq (1 - \epsilon_d^{\text{service}}). \tag{5.4}$$

In a deployment of Arqon, the service error parameter may be chosen according to service contracts (i.e. contracts between the network operator and its clients) or priority classes associated with the demand. In some implementations, it may be set by the central controller.

PERIODIC COMPUTATION AND DISTRIBUTION OF SCHEDULES

The central controller must regularly accept demands, as well as compute and distribute network schedules. Each schedule is associated with a version identifier and covers an

identical execution time known as the *scheduling interval* (denoted T^{SI}). Figure 5.2 is an illustrated overview of the process sequence that enables execution of a network schedule for a scheduling interval k.

In our network architecture there is a constant *scheduling offset*, in terms of a number of scheduling intervals, between the interval at which a demand passes demand registration and the time at which the first schedule involving the demand will be executed if accepted. In Figure 5.2, a scheduling offset of two scheduling intervals is illustrated. To account for this offset, when demands are registered their earliest possible start time $t^{\texttt{start}}$ is calculated and added to the internal representation of the demand.

PACKET GENERATION ATTEMPTS

Network schedules allocate *packet generation attempts* (PGAs) to accepted demands. A PGA always has some duration E, success probability $p^{\texttt{packet}}$, and reserves internal resources on an end-to-end route through the network. PGAs are associated with a success probability 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 PGA.

A minimum separation time between successful packet generation, $t^{\texttt{minsep}}$, is specified as part of a demand. In the PGA framework, this is enforced as a minimum separation time between subsequently scheduled PGAs for the same demand. This minimum separation is included to ensure that there is sufficient time for the local runtime environment on end nodes to execute local operations before the next allocated period of time for generating entangled links begins. Examples of these local operations are additional blocks of quantum operations in the application program, or operations to reset the hardware between subsequent attempts to generate a packet.

PACKET GENERATION TASKS

Packet Generation Tasks (PGTs) are the internal demand representation format in our network architecture [14]. They are persistent tasks requiring the central controller to schedule PGAs to serve an accepted demand until it expires or is terminated. To create a PGT γ from a demand it is necessary to determine a suitable duration E_γ and packet success probability $p_\gamma^{\texttt{packet}}$ for the PGAs. These key parameters of a PGA must be set based on the end-to-end entanglement generation capabilities along a particular path. For this reason, a PGT γ is created for a particular end-to-end entanglement generation path π_γ. We formally define PGTs in Section 5.4.2.

5.4. ARQON SUITE OF CONTROL APPLICATIONS

Arqon consists of four control applications: Network Manager, a Demand Manager, a Network Scheduler, and a Schedule Manager. These control applications are hosted by a central controller, such as an implementation of an Software Defined Network (SDN) controller [77–80]. The central controller manages all communication with other network components. Figure 5.3 is a systems interaction diagram that provides an overview of the control applications and their shared interfaces.

To allow the clearest possible description of the Arqon control applications, we first define the service agreements that Arqon creates when a demand is accepted for service,

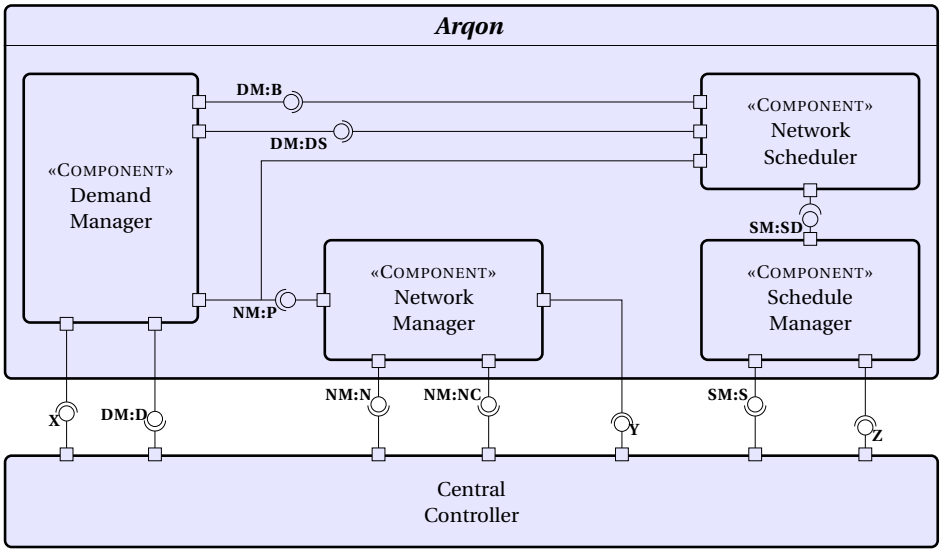


Figure 5.3: System Component diagram for Arqon. The interfaces CC:II are described in the main text. The interface **X** allows Arqon to update the status of demands, interface **Y** allows Arqon to retrieve the current capabilities of network nodes and **Z** allows Arqon to push network schedules to components.

which is accompanied by delivery of an `ACCEPT` message to the end node(s) that submitted the demand.

Definition 37 (Service Agreement)**.** *The service agreement created for an accepted demand is a guarantee from Arqon that:*

"In every scheduling interval between $t^{\texttt{start}}$ and the demand begin terminated or expiring, at least a number of PGAs called a minimal allocation will be scheduled for the PGT accepted to realize the demand."

The minimum allocation is a number $\mathcal{N}_{\gamma_d}^{SI}$ of PGAs that must be scheduled for the PGT γ_d in each scheduling interval between $t^{\texttt{start}}$ and its expiry time, in order to ensure the demand will be satisfied with at least probability $(1-\epsilon_d^{\text{service}})$.

Definition 38 (Minimal allocation)**.** *Let γ_d be a PGT for demand d, let $t^{\texttt{start}}$ be the start time of the first scheduling interval in which demand d may be scheduled, and let $n^{SI} = \lceil (t_d^{\texttt{expiry}} - t_d^{\texttt{start}})/T^{SI} \rceil$. Let $\mathcal{N}_{\gamma_d}^{SI}$ be such that if*

$$X \sim \text{Binomial}(\mathcal{N}_{\gamma_d}^{SI} n^{SI}, p_{\gamma_d}^{\texttt{packet}}), \tag{5.5}$$

then

$$\mathbb{P}[X < N_d^{\texttt{inst}}] < \epsilon_d^{\text{service}}. \tag{5.6}$$

Let $\#_d(S)$ be the number of PGAs for demand d scheduled in schedule S. Then we say a demand receives **minimal allocation** *in schedule S if $\#_d(S) >= \mathcal{N}_{\gamma_d}^{SI}$.*

In the rest of this section we describe each control application in turn, beginning with the Network Manager.

5.4.1. NETWORK MANAGER

The Network Manager is responsible for maintaining an accurate overview of all network components, their relevant capabilities, and current statuses. Its most basic function is to compose and maintain a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, called the *network resource graph*. The vertices $\mathcal{V}$ are network components, and the logical connections between components are indicated by the set of edges $\mathcal{E}$. The Network Manager sets the duration of a scheduling interval, which is a global parameter that must be communicated to the other control applications. The other core processes of the Network Manager are *Node Registration*, *Path Computation* and a *Network Capabilities Manager* (NCM).

The Network Manager provides two interfaces to the central controller, known as the *node* (NM:N) interface and the *network capabilities* (NM:NC) interface, and an internal interface to the other control applications, known as the *path* (NM:P) interface. The node interface is used by nodes (re-)joining the network to register themselves. The network capabilities interface allows end nodes to obtain information about the entanglement generation capabilities of the network. The path interface allows other Arqon control applications to access information about the paths in the network either by requesting a data structure called the path partition, denoted Π, or by requesting sets of paths between specific pairs of nodes.

5

NETWORK RESOURCE GRAPH

The network resource graph, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is an undirected graph in which all the schedulable components in the network are vertices. The set of vertices can be decomposed as $\mathcal{V} = E \sqcup \mathfrak{R}$ where E is the set of end nodes and $\mathfrak{R}$ is the set of schedulable internal resources: EGIs (I), junction nodes (J) and long-distance backbones (B). Each long-distance backbone is allocated as a single unit and is therefor represented by a single vertex in the network resource graph. A network resource graph for a dumbbell network connecting two metropolitan areas across a long-distance backbone is illustrated in Figure 5.1.

The edges of the network resource graph $\mathcal{G}$ represent the logical connections between components. For example, if a metropolitan hub contains two EGIs which can be accessed by end node e, $\mathcal{G}$ has an edge between e and each of these EGIs. In practice, such a setup could be realized with a single physical connection from the end node to an optical switch located at the metropolitan hub. The optical switch could then be configured to open a physical path from the end node to either of the specific EGIs.

A path $\pi = (v_0, v_1, ..., v_k)$ through the network resource graph $\mathcal{G}$ is a sequence of vertices $v_i \in \mathcal{V}$ where for each adjacent pair (v_i, v_{i+1}) there is an edge $e \in \mathcal{E}$ connecting v_i and v_{i+1}, and no vertex is visited more than once. It is only possible to serve demands for end-to-end entanglement generation if there is at least one path through the internal resources connecting two end nodes. If such a path exists between all pairs of end nodes, the network resource graph is internally connected.

Definition 39 (Internally Connected)**.** *We say that a network resource graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with end nodes E is* **internally connected** *if for all pairs of nodes $v, v' \in \mathcal{V}$, there exists a path $\pi = (v, \pi_1, ..., \pi_{k-1}, v')$ in $\mathcal{G}$ such that $\forall i, \pi_i \notin E$.*

End-to-end paths must not pass through end nodes, because these are under local control only and are not bound to execute network schedules.

NODE REGISTRATION

When a node joins the network for the first time, or re-joins the network after a period of inactivity, it needs to register itself with the central controller. This is handled by the Node Registration process of the Network Manager via the node (NM:N) interface with the central controller. When nodes register with the Network Manager it assigns a unique node ID that Arqon uses in the creation and distribution of network schedules, and it triggers updates to the network resource graph and the network capabilities table maintained by the NCM. A registration response is returned to the node with the assigned node ID.

PATH COMPUTATION

Path Computation identifies which paths through the network resource graph are suitable for generating end-to-end entangled links between any two end nodes. It is not a time-critical process and is only triggered by updates to the network resource graph. As end nodes are under local control only, valid end-to-end entanglement generation paths do not traverse end nodes.

Definition 40 (Valid entanglement generation paths)**.** *Let $\mathcal{G}$ be a network resource graph with end nodes E. Let $\pi = (\pi_0, ..., \pi_k)$ be a path in $\mathcal{G}$. Then we say that π is a* **valid entanglement generation path** *if:*

1. $\pi_0 \neq \pi_k$
2. $\pi_0, \pi_k \in E$
3. $\forall i = 1, ..., k-1; \pi_i \notin E$.

We denote the set of all valid entanglement generation paths by $\mathscr{P}_{\text{valid}}$.

The number of valid entanglement generation paths through a network with resource graph $\mathcal{G}$ is $O(2^{|\mathcal{V}|})$. For large networks, this can quickly grow to a size which is infeasible to efficiently compute. Therefore, implementations of Arqon may benefit from reducing the set of valid entanglement generation paths to a set of *allowable paths* by imposing additional restrictions on the paths which may be used to realize a demand.

The outcome of Path Computation is a data structure called the path partition, Π, which is a disjoint partition of all valid or allowable entanglement generation paths. In Section 5.6 we define a specific path partition, but Arqon is compatible with any disjoint partition of $\mathscr{P}_{\text{valid}}$. Looking forward, a disjoint path partition is useful in enabling efficient scheduling of network resources, as non-overlapping paths can be simultaneously scheduled to produce entanglement serving different demands without any ill effects.

NETWORK CAPABILITIES MANAGER

The NCM maintains a *network capabilities table* containing the end-to-end entanglement generation capabilities of each path $\pi \in \Pi$. The network capabilities table has a version identifier that is incremented with each update.

The demands issued by end nodes (see (5.2) and (5.3)) include a minimum fidelity F, a number of instances $N^{\texttt{inst}}$, and an expiry time $t^{\texttt{expiry}}$, all of which are not arbitrary,

but should be informed by the entanglement generation capabilities along paths $\pi \in \Pi$. To obtain this information, end nodes query the central controller for their entries of the network capabilities table [14]. The metadata that is included with a demand includes the network capability version id, which is used by the Demand Manager to screen demands before registering them.

Other Arqon control applications also need to read the entanglement generation capabilities along end-to-end routes, in particular the Demand Manager. For this reason, the path partition Π is updated by the Network Capabilities Manager to specify these capabilities for each path $\pi \in \Pi$.

In practice, the NCM may establish the end-to-end entanglement generation capabilities in a variety of ways. A direct method would be for the central controller to periodically schedule blocks of time for characterization of end-to-end routes. An implementation may include a method of determining the entanglement generation capabilities of an end-to-end route from the capabilities of every link in the route individually.

5.4.2. DEMAND MANAGER

The Demand Manager, is responsible for receiving, screening and registering incoming demands, converting demands into PGTs, maintaining an accurate ledger of demand statuses, and creating the status messages that the central controller sends to end nodes. These processes are implemented by an *Intake and Status Manager*, a *Demand Registration* process, and a *Buffer Manager*. The Demand Manager provides three interfaces: the `DemandStatus` (DM:DS) interface, the `Buffer` (DM:B) interface and the `Demand` (DM:D) interface. The DM:DS interface is used by the Network Scheduler to update demand statuses with `ACCEPT/REJECT` outcomes after admission control. The DM:B interface allows the Network Scheduler to retrieve a buffer of registered demands and a buffer of demands to terminate, held by the Buffer Manager. Finally, the DM:D interface is an interface to the central controller through which the Demand Manager retrieves arriving demands and termination requests for existing demands. The Demand Manager with all internal components and interfaces is illustrated in Figure 5.4.

INTAKE AND STATUS MANAGER

The Intake and Status Manager is the first process which a demand interacts with. For every incoming demand it first assigns a unique demand ID and creates an entry in a ledger of demand statuses, with an initial status flag indicating that the demand is not yet registered. The demand status flag is updated as it progresses through each further processing step. The demand is not removed from the status ledger until it is either rejected, expires, or is terminated by the end nodes. Each incoming demand is added to a First In First Out (FIFO) queue, where it remains until a worker of the Demand Registration process removes it. An implementation may also specify an alternative queuing system for Demand Registration, including a priority based queuing system.

The Intake and Status Manager interprets `PASS/FAIL` outcomes received from Demand Registration and `ACCEPT/REJECT` outcomes received from the Network Scheduler through the DM:DS interface. Upon receipt of a `FAIL` or `REJECT` outcome it issues a demand rejection message to end nodes, with an error code indicating whether the demand failed in Demand Registration (`FAIL`) or was rejected by the Network Scheduler (`REJECT`).

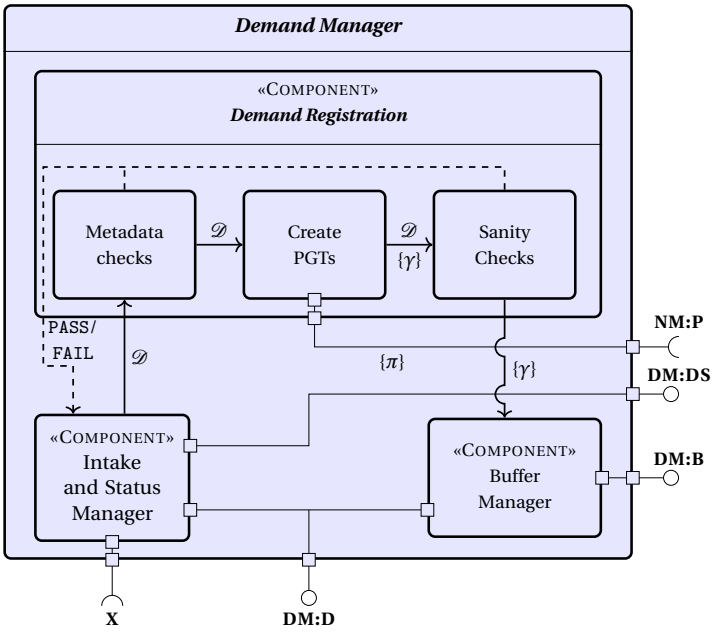


Figure 5.4: Detailed diagram of the internals of the Demand Manager. The arrows show the process of registering a demand, showing how the demand submission $\mathscr{D}$ is passed from metadata checks to Create PGTs. The relevant paths $\{\pi\}$ are obtained via the NM:P interface, before the created PGTs $\{\gamma\}$ are also passed to the sanity checks. Finally, any PGTs which pass the sanity checks are passed to the buffer manager and an `PASS/FAIL` message is sent back to the intake and status manager as appropriate. The interface X provided by the central controller is used to update the statuses of demands at the relevant end nodes.

Upon receipt of a `PASS` or `ACCEPT` outcome, it updates the status of the demand. For an `ACCEPT` outcome the Intake and Status Manager further creates a demand acceptance message for end nodes, which includes a service agreement.

Demand Registration

Demand Registration comprises three phases: a metadata check, the creation of a set of Packet Generation Tasks (PGTs), and implementation specific sanity checks. There are a finite number of parallel instances of the Demand Registration process, known as *workers*, each of which handles the registration of one demand at a time. Workers return a `PASS/FAIL` outcome for each demand to the Intake Manager and take the next demand from the demand queue, as long as it is not empty. An implementation may specify any number of workers, including just a single worker.

1. Metadata check:

   The following assertions must be true for a demand to be valid:

   (a) The network capability version identifier is of the expected data type and corresponds to the latest version;

(b) Both end node id's correspond to registered nodes and are of the expected data type;

(c) There is an allowed/valid path π connecting the source and destination nodes;

(d) The message is a well-formatted demand adhering to the format in (5.2).

(e) Each of $N^{\texttt{inst}}$, F, $w, s > 0$ and all fields are of the expected data type.

(f) The expiry time $t^{\texttt{expiry}}$ is later than $t^{\texttt{start}}$, $t^{\texttt{expiry}} > t^{\texttt{start}}$.

These assertions are necessary to ensure that a demand can be handled in a consistent manner by the network. Additional metadata checks may be specified by an implementation.

2. Create Packet Generation Tasks: A PGT is associated with a particular path from the source to destination end node. To create a PGT γ_d for a demand d with a path $\pi_{\gamma_d} \in \Pi$ it is necessary to set the duration of the PGAs (E_{γ_d}), to calculate the packet success probability ($p_{\gamma_d}^{\texttt{packet}}$), and to calculate the minimum allocation ($\mathcal{N}_{\gamma_d}^{SI}$). In Section 5.6, we specify a particular method of setting E_{γ_d} and calculating $p_{\gamma_d}^{\texttt{packet}}$.

The minimal allocation (Definition 38) is the key parameter of the service agreement that Arqon creates for each accepted demand. It depends on the service error parameter, $\epsilon_d^{\text{service}}$, which is either specified by the end nodes as part of the demand, or set by the central controller.

Definition 41 (Packet Generation Task)**.** *A* **packet generation task** *is a tuple*

$$\gamma = (E_\gamma, \mathcal{N}_\gamma^{SI}, \pi_\gamma, t_\gamma^{\texttt{minsep}}, t_\gamma^{\texttt{expiry}}, t_\gamma^{\texttt{start}}) \tag{5.7}$$

where E_γ is the execution time of each packet generation attempt (PGA), $\mathcal{N}_\gamma^{SI}$ is the minimum allocation for γ, $\pi_\gamma \in \Pi$ is a valid entanglement generation path and $t_\gamma^{\texttt{minsep}}, t_\gamma^{\texttt{expiry}}$ are as in the raw demand (5.2), and $t_\gamma^{\texttt{start}}$ is the first scheduling interval in which a PGA for γ could be scheduled, based on the scheduling offset.

In general, there may be multiple valid entanglement generation paths between two end nodes. Each of these paths is only suitable to realize the demand if its average rate of end-to-end entanglement generation is high enough so that demanded s_d pairs can be generated within a window of time w_d with a non-zero probability $p_{\gamma_d}^{\texttt{packet}} > 0$. In the Create PGTs phase of Demand Registration each demand d is mapped to a set $\mathbf{\Gamma}_d$ of suitable PGTs, from which at most one particular PGT γ will be accepted by the Network Scheduler. This also means that a routing decision for a demand d is not made until the Admit Tasks process of the Network Scheduler, where a particular $\gamma \in \mathbf{\Gamma}_d$ may be accepted for scheduling.

3. Implementation specific sanity checks:

Once $\mathbf{\Gamma}_d$ is constructed, then each possible PGT $\gamma \in \mathbf{\Gamma}_d$ may undergo further sanity checks. If a PGT γ fails a sanity check, then it is removed from $\mathbf{\Gamma}_d$. We define one necessary sanity check, to verify that for each proposed PGT $\gamma \in \mathbf{\Gamma}_d$

$$E_\gamma \leq T^{SI}, \tag{5.8}$$

meaning that the execution time of a single PGA of the PGT γ must not exceed the scheduling interval.

If all PGTs fail the sanity check, then the demand is marked as invalid and a `FAIL` outcome is returned to the Intake Manager. Otherwise, a `PASS` outcome is returned to the Intake Manager and $\mathbf{\Gamma}_d$ is passed to the Buffer Manager to be added to the demand intake buffer, denoted $\mathbf{\Gamma}$.

BUFFER MANAGER

The sets of PGTs $\mathbf{\Gamma}_d$ which can realize each demand d that passes Demand Registration are needed as an input to the Network Scheduler control application. Between Demand Registration and the Network Scheduler however there is a type of timing boundary, in that the Network Scheduler executes a set of periodic processes once per scheduling interval, whereas Demand Registration occurs based on the queue arrival dynamics of demands and the availability of workers. Due to this mismatch in process timings, the set of PGTs γ_d for each registered demand d are placed into a task intake buffer $\mathbf{\Gamma}$, which is maintained by the Buffer Manager.

Within the Buffer Manager, $\mathbf{\Gamma}$ is write only. In contrast, the Demand Manager exposes $\mathbf{\Gamma}$ to the Network Scheduler control application via the DM:B interface, but the buffer is exposed with read only access. The structure used to represent what is read from the task intake buffer by the Network Scheduler is referred to as the *task intake object* and is also denoted $\mathbf{\Gamma}$ for simplicity. After the buffer has been read by the Network Scheduler, the Buffer Manager must implement a flush of the buffer. In an implementation, the Buffer Manager may flush the entire buffer after each read. Alternatively, it may be that the Network Scheduler only reads a maximum number M of sets of PGTs $\mathbf{\Gamma}_i,\ i \in \{0, 1, \cdots, M-1\}$ from the buffer in each read. In that case the Buffer Manager may implement a flush of only the top M sets of PGTs after a read.

Similarly, it needs to maintain a buffer of requests from end nodes to terminate demands, called the *termination buffer* τ. As with $\mathbf{\Gamma}$, this is write-only from the within the Buffer Manager itself, and via the DM:B interface it is exposed in read-only form to the Network Scheduler. Unlike $\mathbf{\Gamma}$, the Network Scheduler should always read the entire termination buffer. Once this has occurred, the entire buffer is flushed so that it is ready to receive more termination requests in the next scheduling interval.

5.4.3. NETWORK SCHEDULER

The Network Scheduler is responsible for determining which PGTs to accept and for producing network schedules that uphold service agreements with end nodes. These network schedules must include a minimum allocation of PGAs for each accepted PGT. It includes an *Admit Tasks* process that examines PGTs $\gamma \in \mathbf{\Gamma}$ in the task intake buffer and determines if they can be accepted without disrupting already accepted PGTs. It also includes a *Compute Schedule* process that occurs after Admit Tasks and produces these network schedules. This control application is time critical, as its processes must start and complete within a single scheduling interval to ensure distribution and execution of a network schedule will not be delayed. Algorithm 6 for NETWORKSCHEDULER is an overview of the main loop of the Network Scheduler control application, and Figure 5.5 illustrates the Network Scheduler with all internal components.

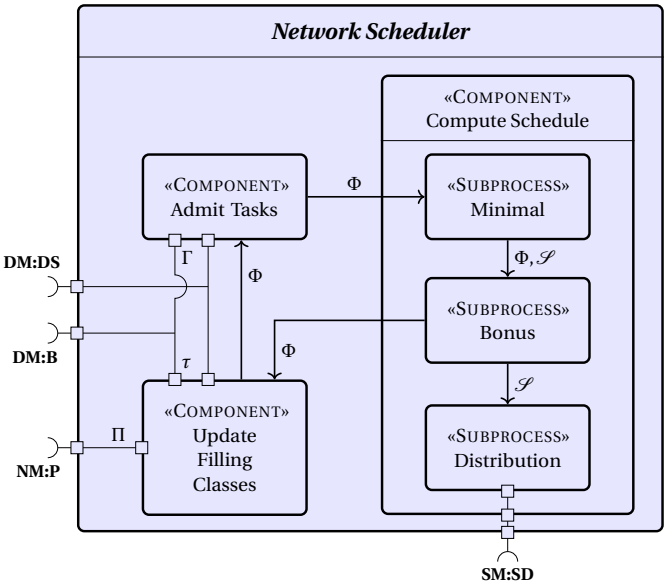


Figure 5.5: Detailed diagram of the Network Scheduler. The arrows show the movement of the set of filling classes Φ and the network schedule $\mathscr{S}$ through the scheduling process.

In our network model, we assume that each network component can only be scheduled to execute PGAs for a single PGT at any given time, and each PGA is non-preemptive. Two different PGAs γ_1 and γ_2 can be scheduled simultaneously as long as $\pi_{\gamma_1} \cap \pi_{\gamma_2} = \emptyset$. To account for these considerations, Arqon structures network schedules as a set of aligned resource schedules, one per network component. To schedule a PGA for a PGT γ, all resources on the path π_γ of the PGT must be simultaneously available for an uninterrupted period of time that is at least as long as the PGA duration E_γ. As illustrated in Figure 5.6, it is possible for a network schedule to contain gaps of time in which certain resources have no PGAs scheduled, but the subsequently scheduled PGAs can not be started earlier due to the requirement that all resources on the path π_γ be simultaneously available. Figure 5.6 also illustrates that there are situations in which the total amount of available time on a network resource is greater than or equal to a PGA duration, however the available time is non-consecutive and therefore the PGA cannot actually be scheduled.

To determine which PGTs can be simultaneously scheduled, Arqon leverages a partition of active PGTs into cells called *filling classes* that are based on the path partition Π. Active PGTs are those that have been accepted in some scheduling interval and have neither expired nor been terminated. The filling classes are used by both Admit Tasks and Compute Schedule.

Definition 42 (Filling Classes)**.** *Let $\Pi = \{\Pi_\phi\}$ be a disjoint partition of a set of paths $\mathscr{P}$ and let Z be a set of PGTs such that $\forall \gamma \in Z$, $\exists \phi$ s.t. $\pi_\gamma \in \Pi_\phi$. The set of* **filling classes** *$\Phi \subset 2^Z \times \Pi$ has elements*

$$\phi = (Z_\phi, \Pi_\phi), \tag{5.9}$$

Algorithm 6: Algorithm for the main process of the Network Scheduler controll application. UPDATEFILLINGCLASSES, ADMITTASKS and COMPUTENEWSCHEDULE are given in Algorithms 9, 10 and 8 respectively.

```
1  Function NETWORKSCHEDULER:
       Input  : Set of filling classes Φ
       Output : List of accepted PGTs A, network schedule S
2      while True:
3          Get Π from the Network Manager ;
4          Get τ, Γ from the Demand Manager;
5          Set Φ, ξ ← UPDATEFILLINGCLASSES(Φ, τ, Π) ;
6          Set Φ, A ← ADMITTASKS(Γ, Φ, ξ);
7          Send (A, Γ \ A) to the Demand Manager;
8          Set S ← COMPUTESCHEDULE(Φ) ;
9          Send S to the Schedule Manager;
10         sleep until the start of the next scheduling interval;
11 end
```

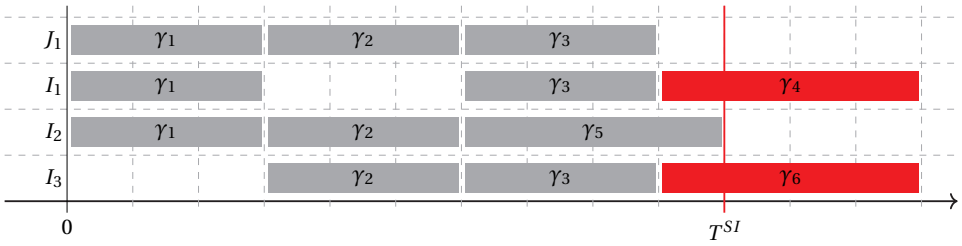


Figure 5.6: Excerpt from a possible network schedule for four components J_1, I_{1-3} and a set of PGTs γ_{1-6}. PGAs for a PGT γ_i are illustrated by gray boxes scheduled simultaneously on every component on their path π_{γ_i}. PGAs for γ_4 and γ_6, illustrated by red boxes, can not be added to the schedule because I_1 (for γ_4) and I_3 (for γ_6) are not available for sufficiently long blocks of uninterrupted time before the end of the scheduling interval, illustrated by the red line labeled T^{SI}.

where $Z_\phi = \{\gamma \in Z : \pi_\gamma \in \Pi_\phi\}$.

The set of paths $\mathscr{P}$ in Definition 42 may be either the set of valid entanglement generation paths $\mathscr{P}_{\text{valid}}$, or a further restriction to a set of allowed paths $\mathscr{P}_{\text{allowed}}$.

Remark 3. *The set of filling classes* Φ *are essentially containers* $\phi \in \Phi$ *for active PGTs, based on the cells of the path partition* $\Pi_\phi \in \Pi$. *The set of filling classes does not need to be populated by PGTs, which means that* Φ *can be created based on the path partition* Π *and an empty set of PGTs* $Z = \emptyset$. *Once created, the set of filling classes may be updated by populating the containers* $\phi \in \Phi$ *with a set of PGTs* $Z \neq \emptyset$.

To identify which resources are required by at least one path in a filling class we introduce the mapping of associated resources between a set of paths and the set of resources on at least one path in the set.

Definition 43 (Associated Resources)**.** *Let $\mathscr{P}$ be a set of paths and let $\mathscr{R}$ be the set of resources which are on some path in $\mathscr{P}$. Let $\{\pi_x\}_x \subset \mathscr{P}$. The* **resources associated** *with $\{\pi_x\}_x$ are indicated by $\xi(\{\pi_x\}_x)$, as defined by the following mapping:*

$$\xi : 2^{\mathscr{P}} \to 2^{\mathscr{R}} \tag{5.10}$$

$$\{\pi_x\}_x \mapsto \{r \in \mathscr{R} : \exists \pi \in \{\pi_x\}_x \; s.t. \; r \in \pi\}. \tag{5.11}$$

UPDATE FILLING CLASSES

In every scheduling interval, the first process that the Network Scheduler does is Update Filling Classes. This process happens first, as both Admit Tasks and Compute Schedule require the updated set of filling classes as input.

The Network Scheduler carries forward the filling classes from a scheduling interval to the subsequent one. However, before proceeding it is necessary to retrieve the path partition from the Network Manager via the NM:P interface and check for a change to the version id. In case of an update to the path partition, the filling classes are rebuilt. There may be one or more PGTs γ for which the path π_γ is no longer in the path partition. In that case, Update Filling Classes must apply a *missing path* rule. The default missing path rule is to remove the PGT and update the demand status to removed, so that the Demand Manager can notify the end nodes and cancel the service agreement. Alternatively, to avoid canceling service agreements, a missing path rule may be implemented that attempts to re-admit an alternative PGT γ' realizing the demand, for which $\pi_{\gamma'}$ is in the updated path partition.

Update Filling Classes is also responsible for removing terminated or expired PGTs. To check for PGT terminations, the Network Scheduler reads the termination buffer τ from the Demand Manager via the DM:B interface. Every PGT is checked and those for which there is a termination message or the expiry time has been reached are removed from their filling class.

The full process Update Filling Classes is described by Algorithm 9, UPDATEFILLING-CLASSES in Appendix 5.10.2.

ADMIT TASKS

The process Admit Tasks is described in full by Algorithm 10, ADMITTASKS in Appendix 5.10.2. It tests PGTs γ' in the task intake buffer $\mathbf{\Gamma}$ to determine whether there is sufficient time available to schedule a minimum allocation $\mathscr{N}_\gamma^{SI}$ of PGAs in subsequent scheduling intervals without disrupting service to any active PGTs. Before testing any new PGTs $\gamma' \in \mathbf{\Gamma}$, it calculates the time required to schedule a minimum allocation of each active PGT and reduces the available time on all resources associated with the PGTs filling class by this duration. The function CALCULATEREQUIREDTIME, defined in Algorithm 7 implements this calculation. Recall that PGAs for a PGT γ must be scheduled for uninterrupted periods of duration E_γ on all components in π_γ. Gaps of time between PGAs of PGTs in the same filling class are not treated as available time by Admit Tasks, because they may not be long enough to allow a PGA to be scheduled. By reducing the available time for all resources associated with a PGTs filling class, rather than just for the resources on the PGTs path, Admit Tasks avoids considering gaps between PGAs as available time. This simplification prioritizes the satisfaction of service agreements, but it may induce an overestimation of the required time and limit the number of accepted PGTs.

Algorithm 7: Calculation of an upper bound on the minimum required time to schedule a minimum allocation of PGAs $\mathcal{N}_\gamma^{SI}$ for every PGT $\gamma \in Z_\phi$ in a filling class ϕ.

Function CALCULATEREQUIREDTIME**:**
  Input : Filling class $\phi = (Z_\phi, \Pi_\phi)$
  Output : Required time for filling class, $R(Z_\phi)$
  Order PGTs $\gamma \in Z_\phi$ by a set of indices $X = \{1, \cdots, |Z_\phi|\}$ such that $\mathcal{N}_x^{SI} \leq \mathcal{N}_{x+1}^{SI}$;
  Set $M \leftarrow |Z_\phi|$;
  Set $n_0^\phi \leftarrow \mathcal{N}_0^{SI} - 1$;
  Set $c_0^\phi \leftarrow \max\left(\max_{y \in \{0, \cdots M-1\}} \left(E_y + t_y^{\texttt{minsep}}\right), \sum_{y=0}^{M-1} E_y\right)$;
  for $x \in \{1, \cdots, M-1\}$ **do**
    Set $n_x^\phi \leftarrow \mathcal{N}_x^{SI} - \mathcal{N}_{x-1}^{SI}$;
    Set $c_x^\phi \leftarrow \max\left(\max_{y \in \{x, \cdots M-1\}} \left(E_y + t_y^{\texttt{minsep}}\right), \sum_{y=x}^{M-1} E_y\right)$;
  end
  Set $R(Z_\phi) \leftarrow \sum_{x=0}^{M-1} (n_x^\phi \cdot c_x^\phi + E_x)$;
  return $R(Z_\phi)$
end

Incoming PGTs in the task intake buffer are organized into sets $\mathbf{\Gamma}_d \in \mathbf{\Gamma}$ by the demand they realize, with each specific PGT $\gamma'_d \in \mathbf{\Gamma}_d$ corresponding to a different path between the end nodes. For each of these PGTs, Admit Tasks repeats the calculation of CALCULATEREQUIREDTIME with γ'_d included in the filling class it would be in (the filling class ϕ such that $\pi_{\gamma'_d} \in \Pi_\phi$). This required time is compared to the duration of the scheduling interval. If it is less than or equal to the scheduling interval the PGT is accepted, the other PGTs in $\mathbf{\Gamma}_d$ are removed and the demand status is set to accepted. Otherwise, the PGT is rejected and Admit Tasks checks the next PGT $\gamma''_d \in \mathbf{\Gamma}_d$, until either one is accepted or no alternatives remain – in which case the demand status is set to rejected. If Admit Tasks accepts a particular PGT to realize a demand, it fixes the path along which the demand will be served. Hence Admit Tasks performs the function of routing.

COMPUTE SCHEDULE

The Network Scheduler computes the network schedule for all components of the network, including end nodes. The schedules of network components are aligned such that a PGA for the PGT γ with path π_γ is added to the schedule of every component on the path π_γ, so that the PGA covers the same span of time in each component schedule. Due to this alignment, it suffices to refer to a single network schedule S, although the schedule is actually a set of individual schedules for each component of the network.

The Compute Schedule process is described by the function COMPUTESCHEDULE in Algorithm 8. It is a time-critical process that must compute a schedule including a minimal allocation of PGAs (Definition 38) for every accepted PGT once per scheduling

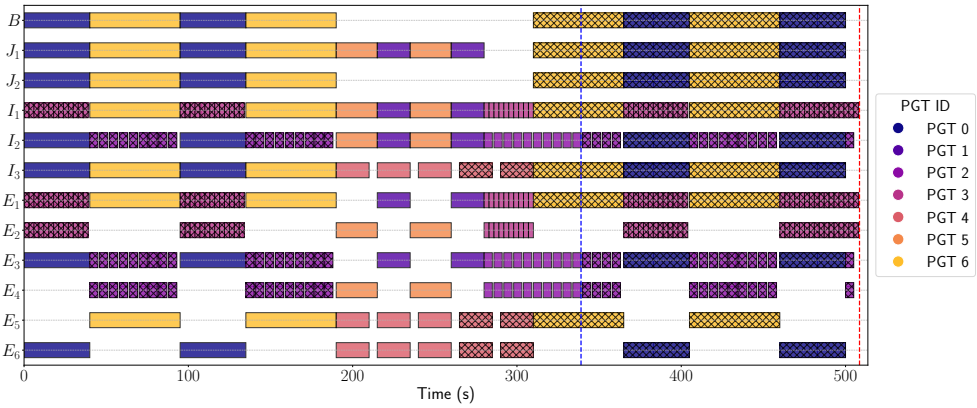


Figure 5.7: An example network schedule produced by COMPUTESCHEDULE for a simplified version of the dumbbell topology network illustrated in Figure 5.1, with a six end nodes and seven active PGTs. End nodes E_{1-2} connect to I_1, E_{3-4} connect to I_2, and E_{5-6} connect to I_3. The PGTs are configured as examples, with different values for their minimal allocations, minimum separations between PGAs, and PGA durations. They are organized into the specific filling classes defined in Section 5.6, so that $Z_{\phi^B} = \{\gamma_0, \gamma_6\}, Z_{\phi^I_1} = \{\gamma_5, \gamma_1\}, Z_{\phi^I_1} = \{\gamma_3\}, Z_{\phi^I_2} = \{\gamma_2\}, Z_{\phi^I_3} = \{\gamma_4\}$. Solid (hatched) color boxes represent PGAs for PGTs of the matching color scheduled by the minimal (bonus) allocation phases of Compute Schedule. The vertical blue dashed line indicates the end of the required time to schedule a minimal allocation of PGAs for each PGT. The red vertical dashed line indicates the end of a scheduling interval.

interval. This is essential for satisfying the service agreements established with accepted demands (Definition 37). Hence, the computation time must not exceed the scheduling interval. A secondary aim of the Compute Schedule process is to minimize the amount of idle time in which no PGAs are scheduled on internal resources. It is divided into two phases: a *minimal allocation* phase which is time-critical and must be completed by the end of a scheduling interval, and a subsequent *bonus allocation* phase which can be cut off at any time without consequences for service agreements.

The minimal allocation phase relies on DIRECTALLOCATION, Algorithm 11 in Appendix 5.10.2, which produces a schedule for one filling class at a time. The DIRECTALLOCATION algorithm is fundamentally based on a modification of the classical Round Robin scheduling algorithm. A schedule S is constructed by calculating a *direct allocation* schedule for each filling class $\phi \in \Phi$ which includes a minimal allocation of PGAs for every PGT $\gamma \in Z_\phi$, and sequentially inserting the direct allocation schedules into S according to a partial ordering of $\phi \in \Phi$. A partial ordering of the filling classes $\phi \in \Phi$ is specified in an implementation and derives from the path partition Π which determines the structure of Φ. The same logic underpins both the DIRECTALLOCATION scheduling algorithm and the CALCULATEREQUIREDTIME function (Algorithm 7) from the Admit Tasks process. CALCULATEREQUIREDTIME essentially calculates the duration of a schedule that would be produced by the DIRECTALLOCATION algorithm, which actually adds PGAs to a network schedule. In this way, Admit Tasks leverages insight into the Compute Schedule process when determining which PGTs to admit.

The bonus allocation phase relies Algorithm 12, ROUNDROBINBONUS of Appendix 5.10.2, which is also based on a modification of the classical Round Robin scheduling algorithm.

Algorithm 8: Overall scheduling algorithm for creating network schedules. DIRECTALLOCATION and ROUNDROBINBONUS are respectively as in Algorithms 11 and 12 in Appendix 5.10.2.

Function MINIMALALLOCATION**:**
  Input : Set of Filling Classes Φ, network schedule S
  Output: Updated network schedule S.
  for $\phi \in \Phi$ **do**
    $S =$ DIRECTALLOCATION$(\phi, S, t_0 + \sum_{\phi' < \phi} R(Z_{\phi'}))$;
  end
end
Function BONUSALLOCATION**:**
  Input : Set of Filling Classes Φ, network schedule S
  Output: Updated network schedule S.
  for $\phi \in \Phi$ **do**
    $S =$ ROUNDROBINBONUS(ϕ, S);
  end
end
Function COMPUTESCHEDULE**:**
  Input : Set of Filling classes Φ
  Output: Compiled Schedule $\tilde{S}$
  Initialise new empty schedule S;
  Set $S \leftarrow$ MINIMALALLOCATION(Φ, S);
  Set $S \leftarrow$ BONUSALLOCATION(Φ, S);
end

In this phase Compute Schedule passes again through the schedule constructed during the minimal allocation phase, and wherever possible schedules additional PGAs. This decreases the idle time of each resource in the network, as well as decreasing the expected time to generate all $N_d^{\texttt{inst}}$ packets of each accepted demand d.

An example network schedule produced by COMPUTESCHEDULE is illustrated in Figure 5.7. This schedule is for a simplified version of the dumbbell topology network illustrated in Figure 5.1, with six end nodes instead of fifteen. There are seven active PGTs in the example, configured to have different values for their minimal allocations, minimum separations between PGAs, and PGA durations. The solid (hatched) colored boxes represent PGAs scheduled during the minimal (bonus) allocation phase. The DIRECTALLOCATION schedule for each filling class has a block like structure and may include gaps of time in which no PGAs are scheduled. These gaps of time carry forward into the minimal allocation part of the schedule S built from concatenated direct allocation schedules. In the bonus allocation phase, additional PGAs are scheduled in some of these gaps, improving the resource utilization and decreasing idle time.

The time complexity of the Compute Schedule process motivates the division of the process into the minimal and bonus allocation phases. In Appendix 5.10.4 we prove that an implementation of the minimal allocation phase has operational complexity

$O(NR) + O(N^2)$, where $N = |Z_\Phi|$ is the number of active PGTs and $R = |\mathfrak{R}|$ is the number of resources in the network. In contrast, the bonus allocation phase has operational complexity $O(N^3 R)$. In the two phase system implemented by COMPUTESCHEDULE, the bonus allocation phase can be cut off by the end of the scheduling interval without impacting demand service agreements. When the scheduling offset is set to two scheduling intervals there is an entire scheduling interval to distribute schedules (as in Figure 5.2) and it is possible to cut off schedule computation arbitrarily close to the end of a scheduling interval.

5.4.4. SCHEDULE MANAGER

The Schedule Manager control application stores the current and subsequent network schedules and formats the component schedules for distribution. It offers two interfaces, the *schedule distribution* (SM:SD) interface and the schedule (SM:S) interface. The SM:SD interface is an internal interface which allows the Network Scheduler to push new schedules to the schedule manager as they are computed. The SM:S interface is an external interface which allows network components to request the current or next schedule. The Schedule Manager consumes the Z interface with the central controller, enabling it to push network schedules for distribution to components.

5.5. PERFORMANCE ANALYSIS

Upholding the service agreement that Arqon establishes for each accepted demand (Definition 37) is essential to ensure the delivery of reliable service. Here we develop the performance analysis of Arqon, by stating and proving results which qualify the operating conditions under which Arqon is guaranteed to satisfy these service agreements. As a supplement to these results, we motivate the connection between satisfaction of service agreements and successful execution of quantum network applications by end nodes. We prove that satisfaction of a service agreement for a demand d with service error parameter $\epsilon_d^{\text{service}}$ translates into end nodes successfully generating all packets of pairs required to execute their targeted application with probability at least $(1 - \epsilon_d^{\text{service}})$.

We define *relevant* scheduling intervals as those that are relevant to the satisfaction of the service agreement.

Definition 44 (Relevant scheduling intervals)**.** *The* **relevant** *scheduling intervals for an accepted demand d are those between $t_d^{\texttt{start}}$ and the scheduling interval in which the demand is either terminated or expires.*

Theorem 13 (Deterministic Satisfaction of Service Agreements)**.** *Let d be a demand accepted by Arqon as the PGT γ_d. Suppose the path partition Π remains static in all relevant scheduling intervals for d and that every component on the path π_{γ_d} retrieves each network schedule before its start time. Then, Arqon deterministically satisfies the service agreement for demand d.*

To prove Theorem 13, which is our main performance result, we first state the following theorem, a proof of which is given in Appendix 5.10.3. The proof of the following theorem is non-trivial and relies upon many intermediate results, each of which is stated and

proved in Appendix 5.10.3. The statement requires the set of filling classes to be *well-behaved*. This property, defined formally in Appendix 5.10.3, means that a method of comparison between filling classes can be defined consistently based on the resources associated with the filling classes.

Theorem 14 (Minimal Allocation is guaranteed)**.** *Let d be a demand accepted by Arqon as the PGT γ_d. Suppose $\pi_{\gamma_d} \in \Pi$ in every relevant scheduling interval. Suppose Φ is a well-behaved set of filling classes. Then, in every relevant scheduling interval the schedule $S = \text{COMPUTESCHEDULE}(\Phi)$ contains at least $\mathcal{N}_{\gamma_d}^{SI}$ PGAs of the PGT γ_d. Moreover, the duration of S does not exceed T^{SI}.*

Proof: Theorem 13. To satisfy the service agreement it is necessary that

1. The schedule is computed before the end of each relevant scheduling interval;
2. The schedule is retrieved successfully by every component on π_{γ_d} in every relevant scheduling interval;
3. The schedule contains at least $\mathcal{N}_{\gamma_d}^{SI}$ PGAs of the PGT γ_d.
4. The duration of the schedule S does not exceed the duration of one scheduling interval T^{SI}.

Property 4. ensures that no schedules overruns the scheduling interval, which would prevent complete execution of the schedule for the subsequent scheduling interval. In the hypothesis, it is assumed that 1. and 2. hold. Properties 3. and 4. are guaranteed by Theorem 14, a full proof of which is given in Appendix 5.10.3. □

An application session succeeds if $N^{\texttt{inst}}$ instances of the quantum application are successfully executed by the end nodes which register it. For this to be possible, at least $N^{\texttt{inst}}$ packets of end-to-end links must be successfully generated. In this way, application success hinges upon the receipt of minimal service.

Definition 45 (Minimal service)**.** *Recall from Definition 36 that a demand d from an application session on end nodes specifies at least $N_d^{\texttt{inst}}$ packets to be generated. We say that an application session achieves* **minimal service** *if at least $N_d^{\texttt{inst}}$ packets of end-to-end links are successfully generated before the demand's expiry time.*

Lemma 4. *Let d be a demand accepted as a PGT γ_d for which Arqon satisfies the service agreement. Suppose all nodes on the path π_{γ_d} attempt all scheduled PGAs. Then, the application session which submitted d achieves minimal service with probability at least $(1 - \epsilon_d^{\text{service}})$.*

Proof. (Lemma 4) By the hypothesis, the service agreement for demand d is satisfied and all nodes on the path π_{γ_d} of the PGT γ realizing the demand attempt all scheduled PGAs. Let x be the total number of packets generated for d as a result of the PGAs attempted. There are two possible scenarios in which a service agreement is satisfied: the demand is either terminated or expires. We are only concerned with demands that expire. If d was terminated, then we may assume that either $N_d^{\texttt{inst}}$ packets were successfully generated

and the application session achieved minimum service, or the application session instance was canceled for some reason unknown to the central controller – in which case the service agreement is also terminated. It follows from the Definition 38 of the minimal allocation that if $\mathcal{N}_{\gamma_d}^{SI}$ PGAs are attempted in each of $n^{SI} = \lceil (t_d^{\texttt{expiry}} - t_d^{\texttt{start}}) / T^{SI} \rceil$ scheduling intervals, then the number x of successfully generated packets satisfies $x \geq N_d^{\texttt{inst}}$ with probability at least $(1 - \epsilon_d^{\text{service}})$. □

5.6. IMPLEMENTATION

We have developed a particular implementation of Arqon written in Python [81] that is designed to enable configurable simulations of realistic networks.

We define a family of network topologies that adheres to a set of physically motivated principles and limit the scope of our implementation to this family of networks. Crucially, our implementation includes the specification of a set of allowed paths $\mathscr{P}_{\text{allowed}}$, a specification of the path partition Π, and the specific choice of filling classes Φ induced by Π. The specific choice of filling classes thus defined has the property of being well-behaved, ensuring compatibility with the performance results in Section 5.5.

To enable numeric evaluations of heterogeneous networks from within this family, with a diverse range of entanglement generation capabilities, we implement a method of generating a network capabilities table which relies on input parameters for mean and standard deviations of the entanglement generation rate and fidelity on fixed length paths, and an scaling calculation based on the length of end-to-end paths.

The full details of our implementation are provided by our open source simulator and its documentation [81]. Here we highlight key features, particularly those that are relevant to the numeric evaluations in Section 4.6.

5.6.1. END NODES

Our implementation is not intended to provide a detailed simulation of end nodes. However, we implement a simple physically motivated model for assigning applications to end nodes and configuring application demands, and we implement demand submission and termination behaviors of end nodes because they are critical for simulating Arqon. In a deployed network, user input will determine which applications an end node runs. We define a set of realistic quantum network applications and implement a method of assigning these applications to end nodes [81].

5.6.2. APPLICATIONS

We define a dictionary of possible applications [**?**] based on minimum fidelity and number of entangled pair requirements of Quantum Key Distribution (QKD) [4, 5], Blind Quantum Computing (BQC) [1, 2], quantum teleportation [82, 83], or any of these applications combined with entanglement purification [84, 85]. We derive the minimum entanglement fidelity and number of pairs requirements from resources [83, 83, 84, 86, 87]. We treat teleportation type applications as *test-applications*, which may be used to validate near term quantum network deployments. For this reason, we simply require an entanglement fidelity of greater than 0.5 for teleportation, which is the classical bound.

ASSIGNING APPLICATIONS

Each application we define in [81] specifies critical information required to produce a valid demand. In particular this includes the (minimum) window duration w. For real end nodes, the maximum memory lifetimes of qubits will affect which application programs can be supported, as these set hard upper bounds for window durations. Application programs don't have an inherent window. In a deployed setting, the local runtime of an end node must set a window based not only on qubit memory lifetimes but also on the duration of time in which a generated packet must remain in memory when executing an instance of the application program. We define a minimum window for an application based on an estimate of how long exemplary quantum hardware may require to execute an application instance. We model the quantum hardware of end nodes based on either the NV center in diamond [39, 88–90] or trapped ion platforms [21, 91–93]. We use data from [39] for the NV platform and from [93] for the trapped ion platform to set the memory lifetimes of individual end nodes to currently experimentally achievable lifetimes in a networked setting (i.e. memory lifetime while simultaneously attempting entanglement generation). Then, when randomly assigning applications to end nodes, we ensure that the memory lifetime of the end node can support the requirements of the application.

In a deployment, some end nodes may be operated as server nodes, for example for serving BQC applications, where less powerful end nodes may be the requesting clients. To support such use cases, we implement a registration of end nodes as either *discoverable* or not. Practically, a server should be discoverable. When assigning applications, we use use this discoverable property as a proxy for a node being a server or a client. We only assign applications between two clients, or between a client and a server.

By mocking these constraints when assigning applications to end nodes, we are effectively replicating some of the behaviors of *capability negotiation*, without fully implementing such a protocol.

DEMAND SUBMISSION

For each application assigned to an end node, we draw the initial demand submission time from a Uniform$[0, t_{\text{rel}}^{\texttt{expiry}})$ distribution, where $t_{\text{rel}}^{\texttt{expiry}} = t^{\texttt{start}} + t^{\texttt{expiry}}$ is the *relative* expiry time. This ensures that the initial demand submissions are not all clustered together around the start of a simulation and mocks the demand submission process by independent end nodes. Thereafter, whenever a demand obtains minimal service (and is then terminated), expires or is removed from the network, a new demand is submitted time Exponential$(t^{\texttt{resubmit}})$ later, where the value of $t^{\texttt{resubmit}}$, the average resubmission time, is set separately for each application source assigned to an end node.

5.6.3. CREATING PACKET GENERATION TASKS

To create a PGT γ_d for a demand d there needs to be a method of calculating E_{γ_d}, the execution time of a PGA for γ_d, $p_{\gamma_d}^{\texttt{packet}}$, the packet generation probability for a PGA of duration E_γ, and $\mathcal{N}_{\gamma_d}^{SI}$, the minimum allocation of PGAs for γ_d. These calculations depend on the demand parameters, the service error parameter $\epsilon_d^{\text{service}}$, and the rate of successful entanglement generation along the path π_{γ_d} of γ_d. In our implementation $\epsilon_d^{\text{service}}$ is a configurable parameter set by the central controller and the same value is used for every demand d.

To calculate possible values of E_{γ_d} for a PGT γ_d we use approximations by Naus from [94] for the probability of k successes in a window w given a rate of average successes λ. For a specified rate of entanglement generation success, these approximations are used to calculate possible pairs E_{γ_d} and $p_{\gamma_d}^{\texttt{packet}}$. To calculate $\mathcal{N}_{\gamma_d}^{SI}$ for a PGT γ_d as in Definition 38 we use Hoeffding's inequality [95]. This calculation depends on the value of $p_{\gamma_d}^{\texttt{packet}}$. See Appendix 5.10.5 for more information about the calculation of E_γ based on $p^{\texttt{packet}}$, and [81] for the Python implementation of these calculations.

We set the final values for E_γ, $p_\gamma^{\texttt{packet}}$, and $\mathcal{N}_\gamma^{SI}$ based on the solution of

$$p_{\gamma_d}^{\texttt{packet}} = \arg\min_p \mathcal{N}_{\gamma_d}^{SI}(p)\left(E_{\gamma_d}(p) + t_{\gamma_d}^{\texttt{minsep}}\right). \tag{5.12}$$

The quantity minimized in (5.12) captures a notion of how much load will be placed on the network by accepting the PGT. The calculation therefore optimizes the selection of $p^{\texttt{packet}}$ for the value that minimizes the load.

5.6.4. NETWORK TOPOLOGY

We specify a set of physically motivated constraints on the network resource graph which define a family of networks. These constraints help to ensure that the network topology of any network we simulate represents a physically realizable quantum network. In a deployment of Arqon these constraints will be replaced by simple input of the actual network topology and the logical connections present in the network. If a deployed network is also an element of our broad family of networks, then it may be equipped with $\mathscr{P}_{\text{allowed}}$ and Π as in our implementation.

NETWORK RESOURCE GRAPH

The first constraint we impose is to consider only internally connected network resource graphs (Definition 39). Every network component $v \in \mathcal{V}$ is uniquely identified as either an end node ($v \in E$), an EGI ($v \in I$), a junction node ($v \in J$), or a long-distance backbone ($v \in B$). Therefore, the set of vertices $\mathcal{V}$ in the network resource graph is a disjoint union of component types,

$$\mathcal{V} = E \sqcup I \sqcup J \sqcup B. \tag{5.13}$$

To further define a family of internally connected networks we introduce a constraint on the $\binom{\mathcal{V}}{2}$ possible edges to a set of allowed edges, denoted $\mathscr{E}^*$. The definition of allowed edges is based on three principles:

1. An EGI is always needed for entanglement generation between two nodes with entanglement generation capabilities [50, 96].

2. Entanglement swapping must always be mediated by a node that has at least one qubit with entanglement generation capabilities and an additional qubit with some memory capabilities [39, 53, 63, 97].

3. Long-distance backbones always own a private set of EGIs, both internally to the long-distance backbone, as well as at the end points where they are exposed to other network components [53, 59–61, 93].

The set of edges $\mathscr{E}$ in a network resource graph must be a subset of the set of allowed edges, $\mathscr{E} \subseteq \mathscr{E}^*$. Informally, the following types of edges are allowed: (e,i), (i,j), (j,b), where $e \in E$, $i \in I$, $j \in J$, $b \in B$ and allowed edges are reflexive, so that (e,i) is an allowed edge means that also (i,e) is an allowed edge. A formal definition of the set $\mathscr{E}^*$ of allowed edges is stated in Appendix 5.10.5. For details on generating random network topologies from within this family of networks, see [81].

Definition 46 (Local Area)**.** *Let $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ be a network resource graph, and B the set of long-distance backbones. Let $\mathscr{E}_B = \{(e_1, e_2) \in \mathscr{E} \mid e_1, e_2 \in B\}$ and $\overline{\mathscr{E}_B} = \mathscr{E} \setminus \mathscr{E}_B$. Let*

$$\overline{\mathscr{G}_B} := (\mathscr{V} \setminus B, \overline{\mathscr{E}_B}) \tag{5.14}$$

be the subgraph of $\mathscr{G}$ formed by removing every long-distance backbone vertex and corresponding edges. Then the **local areas** *of $\mathscr{G}$ are the (maximal) internally connected sub-components of $\overline{\mathscr{G}_B}$. The set of local areas is denoted $\mathscr{L}$.*

The concept of a local area is introduced to aid in describing the scale of a quantum network. Local areas induce a structure on the network resource graph such that it can be decomposed into the graph $\overline{\mathscr{G}_B}$ and a graph $\mathscr{G}_C$ which contains only local areas and long distance backbones. The graph $\mathscr{G}_C$ captures the local area connectivity of $\mathscr{G}$.

5.6.5. Path Computation

The set of allowed paths $\mathscr{P}_{\text{allowed}} \subset \mathscr{P}_{\text{valid}}$ is a restriction on the set of valid entanglement generation paths. In Appendix 5.10.5 we formally define $\mathscr{P}_{\text{allowed}}$ following a construction which minimizes the number of long-distance backbones traversed, rather than strictly minimizing the absolute number of hops in a path. This construction is motivated by the expectation that a relatively large loss of entanglement generation success rate will result from traversing of a long-distance backbone, as compared to traversing multiple hops within a local area [61, 93, 98].

Definition 47 (Specific choice of path partition)**.** *Let $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ be an internally connected resource graph and let $\mathscr{E} \subseteq \mathscr{E}^*$. Let $\mathscr{L} = \{L_i = (\mathscr{V}_i, \mathscr{E}_i)\}$ be the set of local areas of $\mathscr{G}$. Let $J_i = J \cap \mathscr{V}_i$ be the junction nodes in sub-graph L_i. Denote by $\pi = (\pi_1, \pi_2, \cdots)$ a path π with vertices $\pi_i \in \mathscr{V}$.*

We define the partition Π of $\mathscr{P}_{\text{allowed}}$ in the following manner:

$$\Pi^B = \{\pi \in \mathscr{P}_{\text{allowed}} : \exists k \; s.t. \; \pi_k \in B\}$$
$$\Pi^J_j = \{\pi \in \mathscr{P}_{\text{allowed}} \setminus \Pi_b : \exists k \; s.t. \; \pi_k \in J_j\}, \; \forall \leq j \in J$$
$$\Pi^I_i = \{\pi \in \mathscr{P}_{\text{allowed}} \setminus (\Pi_b \cup \bigcup_{j \in J} \Pi^J_j) : \exists k \; s.t. \; \pi_k = i\}, \quad \forall i \in I,$$

where $\mathscr{P}_{\text{allowed}}$ is as defined Appendix 5.10.5, Definition 58.

Lemma 5.

$$\Pi = \{\Pi^B\} \cup \{\Pi_j\}_{j \in J} \cup \{\Pi_i\}_{i \in I}, \tag{5.15}$$

where Π^B, $\Pi^J_j \; \forall \, j$, and $\Pi^I_i \; \forall \, i$ are as in Definition 47, is a disjoint partition of $\mathscr{P}_{\text{allowed}}$.

Proof. See Appendix 5.10.5. □

5.6.6. NETWORK CAPABILITIES

The network capabilities manager populates the network capabilities table using a simple heuristic model when the network is initialized. Our implementation does not include dynamic update handling for the capabilities of network components or paths through the network. See [81] for full details of the model.

5.6.7. FILLING CLASSES

The properties of the set of filling classes are directly inherited from the specific choice of path partition, which defines their structure (Definition 47). The specific choice of filling classes can be written as

$$\Phi = \{\Phi^B\} \cup \{\Phi^J_j\}_{j \in J} \cup \{\Phi^I_i\}_{i \in I}. \tag{5.16}$$

In Appendix 5.10.5 we prove that this specific choice of filling classes has the property of being well-behaved.

5.6.8. EXECUTION OF NETWORK SCHEDULES

Since we do not implement a detailed model of end nodes, in our implementation we mock the execution of network schedules by sampling random variables to determine how many PGAs successfully generate packets. We then assume that whenever a packet is generated as the result of a PGA, then the corresponding application instance is successfully executed. Our implementation does include session termination behavior in which end nodes terminate application sessions after they have obtained minimal service.

For any network schedule in which a PGT γ cannot reach the termination criteria of obtaining minimal service (i.e. less PGAs are scheduled than the remaining instances to execute), we determine the number of packets generated by drawing from a Binomial distribution with probability of success $p_\gamma^{\texttt{packet}}$. Otherwise, we sample from a Bernoulli distribution with probability of success $p_\gamma^{\texttt{packet}}$ for each PGA in turn. This two-stage approach allows us to minimize the time required to simulate execution of network schedules, while still allowing accurate reporting of when minimal service is obtained.

5.7. COMPLEXITY ANALYSIS

Arqon must produce a new network schedule every scheduling interval, with a total computation time not exceeding the scheduling interval. To validate that this is feasible, we analyze the operational complexity of our implementation of the Network Scheduler control application. Our operational complexity analysis is based on determining which fundamental operations, such as value assignment, list insertion, list appending, list sorting, etc. occur in the implementations of our algorithms, as in Listings 5.1-5.8 in Appendix 5.10.6. For each fundamental operation, such as value assignment, we determine the time complexity of the operation, for example: value assignment is O(1). We therefore calculate the time complexity of our algorithms based on the number of operations of each type, and the time complexity of each type of operation. The details of our analysis are given in Appendix 5.10.4.

Theorem 15. *The complete program* NETWORKSCHEDULER *has operational complexity*

$$O\Big(k2^R\big((N+k)^2+R\big)+(N+R)^2+N^3R\Big), \tag{5.17}$$

where $R = |\mathfrak{R}|$ is the number of internal resources in the network, $N = |Z_\Phi|$ is the number of active PGTs, and $k = |\mathbf{\Gamma}|$ is the number of PGTs in the task intake object.

Proof. The program NETWORKSCHEDULER consists of the program sequence UPDATEFILLINGCLASSES, ADMITTASKS, and COMPUTESCHEDULE. By Lemma 14 UPDATEFILLINGCLASSES has operational complexity

$$O((N+R)^2). \tag{5.18}$$

By Theorem 19 ADMITTASKS has operational complexity

$$O\Big(k2^R\big((N+k)^2+R\big)\Big). \tag{5.19}$$

The program COMPUTESCHEDULE consists of a minimal allocation phase, a bonus allocation phase, and a schedule compilation phase. The minimal allocation phase contributes (Lemma 17)

$$O(NR)+O(N^2) \tag{5.20}$$

to the operational complexity. The bonus allocation phase contributes (Lemma 18)

$$O(N^3R). \tag{5.21}$$

Compiling the resulting schedule contributes only $O(NR)$ (Lemma 19). The total operational complexity of COMPUTESCHEDULE is therefore dominated by the bonus allocation phase and is given by Theorem 20 as

$$O(N^3R). \tag{5.22}$$

Overall, the program NETWORKSCHEDULER has operational complexity,

$$O\Big(k2^R\big((N+k)^2+R\big)+(N+R)^2\Big)+O(N^3R).$$

□

Our implementation of the Network Scheduler has an operational time complexity with a polynomial dependency on the number of active PGTs N and the size of the task intake buffer k. However, there is an exponential dependence on the number of resources in the network, R. In a deployment, the number of resources is expected to be relatively static, and will typically act as a constant factor in the operational complexity of the Network Scheduler. Nevertheless, the exponential dependence on R does indicate that Arqon is best suited for networks in which $R < 1000$. This result conforms with the heuristic expectation that distributed control architectures are more suitable than centrally controlled network architectures for very large networks [99].

5.8. EVALUATION

Arqon is designed to deliver reliable service, according to reliability requirements *(R1)*-*(R4)*. The first requirement, *(R1)* is satisfied by design, as all demands receive accept/reject messages from the demand manager within two scheduling intervals following their registration [81]. By Theorem 15, which captures the overall operational complexity of the Network Scheduler control application, it is expected that network schedules can always be computed in time to be delivered. Then, by Theorem 13, it is expected that Arqon satisfies requirements *(R2)*-*(R4)* as long as the path partition Π remains static throughout an evaluation. However, from the perspective of a network operator, there remain several relevant questions regarding the manner in which reliable service is delivered, including the following.

(Q1) What effect does the service error parameter $\epsilon^{\text{service}}$ have on the service delivered?

(Q2) How quickly does Arqon satisfy service agreements, relative to demand expiry times?

(Q3) What impact does the bonus allocation phase of the Compute Schedule process of the Network Scheduler have on the service delivered?

(Q4) What proportion of submitted demands are accepted?

(Q5) What impact does the specific choice of filling classes (5.16) have on which of the registered PGTs are accepted by Admit Tasks?

In this section we first validate that Arqon delivers reliable service on static network topologies and then address these questions, which capture additional performance metrics. Finally, we validate our operational complexity analysis of the Network Scheduler with thorough numeric evaluations that supply concrete time values for computation times in a variety of parameter regimes.

In all simulations, the demands for service submitted by pairs of end nodes are based on the real minimum fidelity requirements of BQC (2, 6, or 10 qubit) [83, 87], e91 based QKD [86], teleportation [83], or any of these applications pre-pended by purification following the DEJMPS protocol [84] of the bipartite entangled pairs. The other parameters of each demand, such as the number of pairs and the window duration are set based on the application and the network capabilities database for that simulation [81]. Lists of the applications simulated in each simulation are included as JSON files in [81].

5.8.1. SERVICE RELIABILITY

To meet the reliability requirements *(R2)*-*(R4)*, Arqon must satisfy demands before their deadlines and the arrival of new demands must not disrupt service to already accepted demands. A demand is satisfied when its associated application session achieves minimal service (Definition 45). By Lemma 4, if the service agreement for a demand d is satisfied, then the corresponding application session achieves minimal service with at least probability $(1-\epsilon_d^{\text{service}})$.

We validate that Arqon delivers reliable service first with simulations of a network with the dumbbell topology illustrated in Figure 5.1, and with $\epsilon^{\text{service}} = 1e-5$ for all

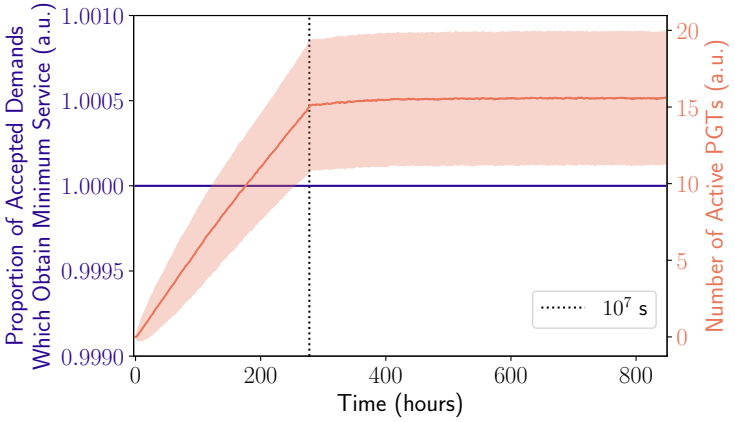


Figure 5.8: Proportion of accepted demands which obtain minimal service (left, blue) and the number of active PGTs (right, orange), compared to the time at which the scheduling interval occurs. The scheduling interval duration 30 minutes and the data corresponds to a simulated 10 weeks of continuous network operation. Shaded regions show 1 standard deviation of the mean.

demands. Then, we validate that the reliability of service delivery is not conditional on a specific network topology or number of demand sources by simulating random network topologies from the family of networks defined in Section 5.6.

DUMBBELL TOPOLOGY NETWORK

We simulate one thousand randomly seeded runs, each corresponding to 846 hours (1694 half hour scheduling intervals) of continuous operation of the dumbbell topology network in Figure 5.1. In every run, applications are assigned to 80% of all possible end node pairings connected by a route in $\Pi_i^I \ \forall i \in I$, 10% of all end node pairings connected by a route in $\Pi_j^J \ \forall j \in J$, and 10% of all end node pairings connected by a route in Π^B. As long as at least one of the nodes assigned the application is 'not discoverable' (see Section 5.6), then the application session submits demands for service. These simulations included an average of 18 demand sources, with the (static) number per simulation ranging from 3 to 26.

Figure 5.8 shows how two metrics vary over time for the dumbbell topology simulations: (1) the proportion of accepted demands that obtain minimal service, and (2) the number of active PGTs at any given time. For the minimal service proportion metric, time corresponds to when each demand was accepted as a particular PGT. Across all one thousand simulation runs, a total of 180,662 accepted demands reached their expiry times. With the stringent value of $\epsilon^{\text{service}} = 1e-5$ used in these simulations, Lemma 4 predicts that at most 2 accepted demands should fail to obtain minimal service. The simulation results confirm this expected behavior: all accepted demands obtained minimal service (proportion = 1.0, standard deviation = 0).These outcomes validate that the service delivered satisfies reliability requirements *(R2)* and *(R3)*.

The variation in the number of active PGTs over time enables validating that arriving demands do not disrupt service to already accepted demands *(R4)*. Starting from network

Long-distance Backbones	Local Areas	End Nodes
1	2	15
2	2	15
2	2	50
2	3	30
2	3	50
3	3	30
5	4	40
6	3	35
7	5	50
12	4	40

Table 5.1: Parametrization of the randomly generated topologies simulated.

initialization at time zero, pairs of end nodes begin submitting demands and some of these are accepted as specific PGTs by Admit Tasks. In these simulations, the time at which different end node pairs submit their first demand is uniformly distributed over the first 10^7 seconds of network operation. This process results in the number of PGTs to schedule increasing over time, before reaching a steady state average of 15.5 PGTs after about 278 hours (10^7 seconds) of continuous operation. The ramp-up in the number of PGTs to schedule has no impact on the proportion of demands obtaining minimal service, confirming that *(R4)* is satisfied.

RANDOM NETWORK TOPOLOGIES

The random topologies we simulate are parameterized in Table 5.1 by the number of long-distance backbones, the number of local areas, and the number of end nodes in the network. For each parameterization, we randomly generate ten network topologies and then, we simulate ten randomly seeded runs for every topology. Every run corresponds to 757 hours (1514 half hour scheduling intervals) of continuous operation. In every run we assign application sessions to 20% of all possible end node pairings connected by a route in $\Pi_i^I \ \forall i \in I$, 15% of end node pairings connected by a route in $\Pi_j^J \ \forall j \in J$, and 5% of end node pairings connected by a route in Π^B.

To address *(Q1)* and study the impact of different values for the $\epsilon^{\text{service}}$ parameter, we perform each simulation with five different settings:

$$\epsilon^{\text{service}} \in \{1e-5,\ 0.001,\ 0.01,\ 0.1,\ 0.5\}.$$

In Table 5.2 we record the proportion of accepted demands which obtain minimal service, which is always greater than the lower bound of $(1-\epsilon^{\text{service}})$ predicted by Lemma 4. These results validate that Arqon delivers reliable service for a wide range of network topologies and number of demand sources.

5.8.2. Additional Performance Metrics

The average values of performance metrics addressing *(Q1)*-*(Q3)* are reported in Table 5.2. To characterize the impact of the bonus allocation phase of Compute Schedule, we disabled this feature and repeated all simulations of randomly generated network topologies with the same random seeds as in the simulations with default Arqon. The total proportion of accepted demands which obtain minimal service in these simulations was always $> 99\%$, both with and without the bonus allocation phase. This outcome persisted even when $\epsilon^{\text{service}}$ was relaxed to the very high value of 0.5, which Lemma 4 predicts to allow up to 50% of demands to fail to obtain minimal service.

These results indicate that Arqon over-serves accepted demands, as compared to the service error parameter $\epsilon^{\text{service}}$. As this effect persists when the bonus allocation phase is disabled, we trace the cause of this excess service to how Arqon calculates the minimal allocation for each demand (Definition 38), which has proven to be a large overestimate. Application sessions on end nodes may benefit from over-service of accepted demands, but wider network performance metrics such as the proportion of accepted demands may suffer from over-service to demands. In an alternate implementation, this may be addressed by replacing the method which solves the the system of equations determining the minimal allocation with a weaker solution that determines smaller values for the minimal allocation.

Arqon schedules a minimal allocation of PGAs for all active PGTs in every scheduling interval, regardless of whether or not the expected number of successfully generated packets,

$$\mathbb{E}[\#\text{ successful PGAs}] = N^{\text{PGAs scheduled}} \times p^{\texttt{packet}} \tag{5.23}$$

already exceeds the number of required successes ($N^{\texttt{inst}}$). To prevent over-serving demands, an alternate implementation may decrease the minimal allocation for PGTs following the scheduling interval in which (5.23) exceeds $N^{\texttt{inst}}$.

The data in Table 5.2 also addresses *(Q2)* and *(Q3)*.

(Q2): Arqon satisfies service agreements for accepted demands significantly in advance of demand expiry times. This is true for all $\epsilon^{\text{service}}$ values simulated and remains true if the bonus allocation phase of Compute Schedule is disabled.

(Q3): The bonus allocation phase significantly impacts how quickly an accepted demand obtains minimum service, relative to its expiry time. The mean service to expiry time more than doubles when the bonus allocation phase is disabled.

Approximately half of all PGAs scheduled by Arqon are scheduled by the bonus allocation phase of Compute Schedule, regardless of the value of $\epsilon^{\text{service}}$. This proportion directly accounts for the speedup of the service to expiry time introduced by this phase.

Proportion of Demands Accepted

The proportion of submitted demands that are accepted, the subject of *(Q4)*, is a performance metric that captures information about the load on a network relative to the tolerance of Arqon's admission control algorithm (Admit Tasks). It is a metric that directly impacts application sessions on end nodes, as a low proportion of accepted demands

$\epsilon^{\text{service}}$	Mean Proportion of Accepted Demands Which Obtain Minimal Service	Mean Service to Expiry Time	Mean Proportion of PGAs Scheduled in Bonus Allocation
Default Arqon: Bonus Allocation Enabled in Compute Schedule			
1e-05	1.0000	0.3706	0.4950
0.001	1.0000	0.3777	0.5279
0.01	1.0000	0.3809	0.5382
0.1	1.0000	0.3847	0.5386
0.5	0.9998	0.3837	0.5423
with Bonus Allocation Disabled in Compute Schedule			
1e-05	1.0000	0.7481	0
0.001	1.0000	0.8194	0
0.01	1.0000	0.8483	0
0.1	0.9998	0.8639	0
0.5	0.9956	0.8646	0

Table 5.2: Additional performance metrics addressing *(Q1)*-*(Q3)* from simulations of the randomly generated network topologies parameterized in Table 5.1.

may translate into long wait times to receive a service agreement, a result of demands needing to be re-submitted multiple times.

For simulations of the Dumbbell topology network, Figure 5.9 plots the average proportion of submitted demands that are accepted by Admit Tasks per scheduling interval and the number of active PGTs in each scheduling interval. The proportion of demands accepted as some particular PGT decreases as a function of the number of active PGTs. It reaches a steady state value of 0.15 after 10^7 s, after which time all application sessions have submitted at least one demand. When demands are rejected, terminated, or expire, application sessions on end nodes submit new demands following a Poisson process with a session specific resubmission rate.

For the simulations of randomly generated network topologies, the average proportion of demands accepted and the proportion of demands which pass demand registration, both with default Arqon and with the bonus allocation phase of Compute Schedule disabled are recorded in Table 5.3. With default Arqon, for all values of $\epsilon^{\text{service}}$ simulated the average proportion of demands accepted was between 0.12 and 0.22. In contrast, the proportion of demands which passed Demand Registration always exceeded 0.98. Therefore, almost all rejected demands are rejected by Admit Tasks rather than be Demand Registration. With bonus allocation disabled, the average proportion of demands accepted decreased by at least 0.036 for every simulated value of $\epsilon^{\text{service}}$, and by up to 0.073 for $\epsilon^{\text{service}} = 0.5$.

For both the dumbbell topology simulations and the simulations of random network topologies, the average proportion of demands accepted is low, indicating that the average load on the network in our simulations is similar to the maximum load that the Admit Tasks calculates can be supported. Combining with the results from Table 5.2 that

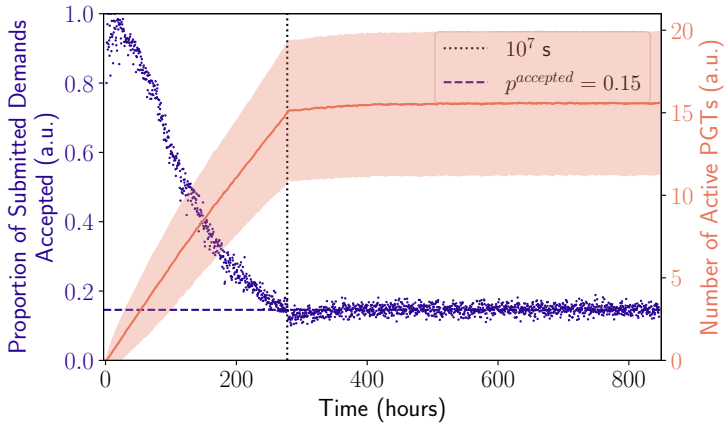


Figure 5.9: Proportion of submitted demands which are accepted at a specific time (left, blue) and the number of active PGTs (right, orange) over time. Application sessions are first created at times that are uniformly distributed over the first 10^7 s of network operation, emphasized by the vertical black dotted line. The horizontal dashed blue line indicates the average of the points after the black dotted line.

indicate Arqon over-serves accepted demands, we expect that the acceptance criterion of Admit Tasks is more stringent than is necessary to satisfy reliability requirement *(R4)*. An alternate implementation of Arqon may modify the function CALCULATEREQUIREDTIME, with which AdmitTasks calculates the time required to schedule a minimal allocation of all active PGTs. A possible modification would be to omit from the calculation any accepted PGTs γ for which the expected number of successfully generated packets, given by (5.23), already exceeds the required number, $N_\gamma^{\texttt{inst}}$.

PROFILING ACCEPTED PGTS

To address *(Q5)* Figure 5.10 plots the number of submitted PGTs (Figure 5.10a) and the number of accepted PGTs (Figure 5.10b) in each filling class. There is one data point for each scheduling interval, for each network topology parameterization in Table 5.1. Data points are averages over the ten random topologies generated from each parameterization and the randomly seeded simulations for each topology.

The number of active PGTs in one filling class can impact the proportion of PGTs admitted to another filling class because Admit Tasks reserves time for active PGTs on the resources associated with their filling classes. In our implementation of Arqon, we employ the specific choice of filling classes (5.16) derived from the path partition in Definition 47. In this path partition, all paths which contain at least one long-distance backbone are in the same cell of the partition. For network topologies with a large number of local areas and long-distance backbones, the result is that PGTs with paths between only two local areas are in the same filling class as PGTs with paths between multiple local areas.

In Figure 5.10 there is some clustering of the registered PGTs (Figure 5.10a) based on the number of local areas in the network parameterization. In contrast, after Admit Tasks (Figure 5.10b) there is clear clustering of the active PGTs based on the unique network topology parameterization, with ten data groupings visible in each subfigure, each corresponding to one of the ten distinct network parameterizations.

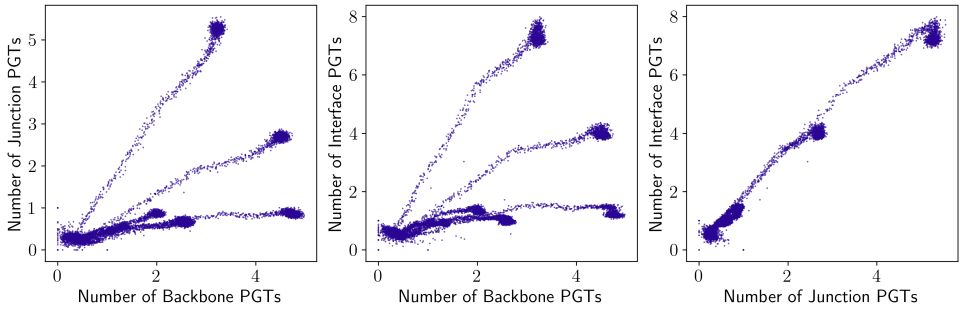
$\epsilon_{\text{service}}$	Total proportion of Demands Accepted	Total Proportion of Registered Demands Accepted	Total Proportion of Demands which Pass Demand Registration
Default Arqon: Bonus Allocation Enabled in Compute Schedule			
1e-05	0.127	0.131	0.992
0.001	0.123	0.128	0.991
0.01	0.102	0.105	0.993
0.1	0.120	0.123	0.993
0.5	0.211	0.222	0.984
with Bonus Allocation Disabled in Compute Schedule			
1e-05	0.073	0.077	0.992
0.001	0.087	0.096	0.986
0.01	0.042	0.047	0.991
0.1	0.055	0.069	0.982
0.5	0.138	0.155	0.977

Table 5.3: The total proportion of demands registered and accepted as particular PGTs, addressing *(Q4)*, in simulations of the randomly generated network topologies parametrized in Table 5.1.

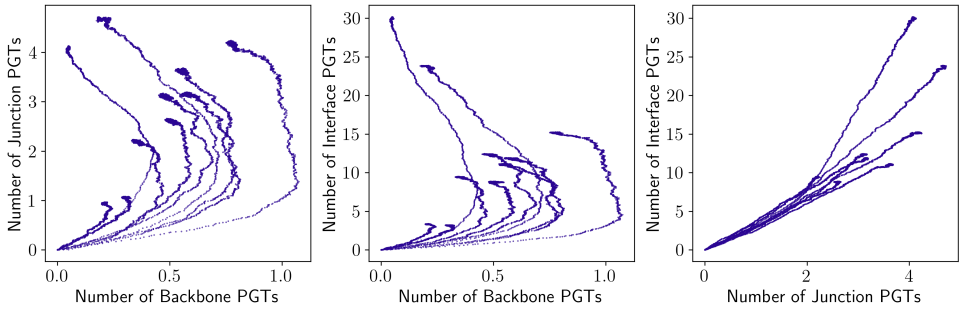
For all network parameterizations simulated, the number of registered PGTs in a junction ($\Phi_j^J \ \forall j \in J$) or interface ($\Phi_i^I \ \forall i \in I$) filling class increases nearly monotonically with the number of registered PGTs in the backbone filling class (Figure 5.10a). Similarly, the number of registered PGTs in an interface filling class increases nearly monotonically with the number of registered PGTs in a junction filling class. The relative number of PGTs registered in each filling class derives immediately from the simulation configuration, which sets the number of application sessions between end nodes connected by routes in interface, junction, and backbone filling classes. After Admit Tasks (Figure 5.10b), there remains a nearly monotonic increase in the number of active PGTs in an interface filling class with the number of accepted PGTs in a junction filling class. In contrast, the numbers of active PGTs in a junction or interface filling class relative to the number in the backbone filling class indicate a different relationship. The number of PGTs accepted to the backbone filling class inhibit acceptance of PGTs to both junction and interface filling classes. To combat this effect an alternate implementation of Arqon may define a path partition which sub-divides the backbone cell of the partition Π^B based on the number of local areas traversed by a path.

5.8.3. NUMERICAL COMPLEXITY ANALYSIS

In every simulation conducted, Arqon was able to compute the network schedule much faster than the cutoff time for schedule distribution ($t_{\text{compute}} \ll T^{SI}$). The average time to produce a network schedule for each of the random topologies simulated is recorded in Table 5.5 in Appendix 5.10.4. The maximum of the average times required to produce a network schedule was less than 0.25 seconds (t_{compute}), whereas the duration of the scheduling interval was 1800 seconds (T^{SI}).



(a) Number of Registered PGTs



(b) Number of Active PGTs

Figure 5.10: Profiling of PGTs registered by Demand Registration and active PGTs accepted by Admit Tasks based on their filling classes [backbone (Φ^B), junction ($\Phi^J_j \ \forall j \in J$), and interface ($\Phi^I_i \ \forall i \in I$) filling classes]. There is one data point for each scheduling interval, for each network topology parameterization in Table 5.1. Data is averaged over the ten random topologies generated from each parameterization, and ten randomly seeded simulations for each topology.

In Section 5.7 we proved Theorem 15, which states that the full program NETWORKSCHEDULER has operational complexity that is polynomial in the number of PGTs in the task intake buffer ($k = |\mathbf{\Gamma}|$) and the number of active PGTs ($N = |Z_\Phi|$), but exponential in the total number of internal resources in the network ($R = |\mathfrak{R}|$). We validate our complexity analysis with numerical results, demonstrating how operational complexity translates to execution time for various combinations of k, N, and R. First, we assess the computation time of ADMITTASKS using randomly generated sets of active PGTs (Z_Φ and PGTs in the task intake buffer ($\mathbf{\Gamma}$). The methods for generating these random sets are included in our simulator [81]. Second, we evaluate COMPUTESCHEDULE with a carefully designed set of PGTs to schedule (Z^*_Φ) for which specific algorithm characteristics produce clear features in the computation time. This second evaluation complements our earlier simulations on random network topologies, where schedule computation times were consistently aligned with the complexity analysis.

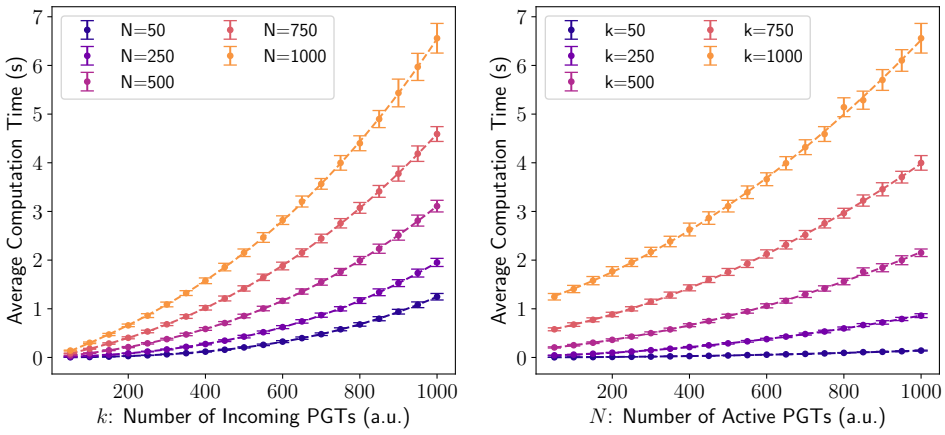


Figure 5.11: Average computation time of Admit Tasks for $N = |Z_\Phi|$ active PGTs and $k = |\mathbf{\Gamma}|$ incoming PGTs requiring an accept/reject decision from Admit Tasks. Each data point is the average of 100 randomly generated $Z_\Phi, \mathbf{\Gamma}$ pairs and error bars show one standard deviation. Data points are fitted with polynomials of $O(k^3)$ (left) and $O(N^2)$ (right).

5

ADMIT TASKS

The operational complexity of ADMITTASKS is given by (5.19). Figure 5.11 plots the computation time against k for constant N (left) and against N, for constant k (right). Each data point is the average over one hundred randomly generated Z_Φ and $\mathbf{\Gamma}$ pairs, with error bars indicating one standard deviation. Polynomial fits of $O(k^3)$ (left) and $O(N^2)$ (right) confirm scaling consistent with (5.19), validating Theorem 19. Importantly, these results demonstrate that ADMITTASKS is fast enough to process a large number of incoming PGTs (k) within a single scheduling interval, even when there are many active PGTs (N). For example, one thousand incoming PGTs ($k = 1000$) are processed in less than seven seconds when there are already one thousand active PGTs ($N = 1000$).

COMPUTE SCHEDULE

The operational complexity of COMPUTESCHEDULE is given by (5.22). The minimal allocation phase, implemented by Algorithm 11, contributes $O(NR) + O(N^2)$ and contains no branching. The total number of operations depends on N, but not on particular PGT characteristics. The resulting network schedule may contain gaps during which no PGAs are scheduled, as illustrated in Figure 5.7. The bonus allocation phase, implemented by Algorithm 12 attempts to fill these gaps. It contributes $O(N^3R)$, which dominates the overall complexity. Its branching behavior depends non-trivially on the number and duration of gaps, the execution times E_γ ($\forall \gamma \in Z_\Phi$) of PGAs that may be added, and the minimum separation $t_\gamma^{\texttt{minsep}}$ ($\forall \gamma \in Z_\Phi$) between PGAs of each PGT.

To illustrate this branching behavior, we consider a specific set of PGTs (Z_Φ^*) with the following structure:

- the first PGT γ_0 has $t_{\gamma_0}^{\texttt{minsep}} = 200$ seconds, and all other PGTs γ_i, $i > 0$ have $t_{\gamma_i}^{\texttt{minsep}} = 0$;
- $E_\gamma = 1$ second, $\forall \gamma \in Z_\Phi^*$;

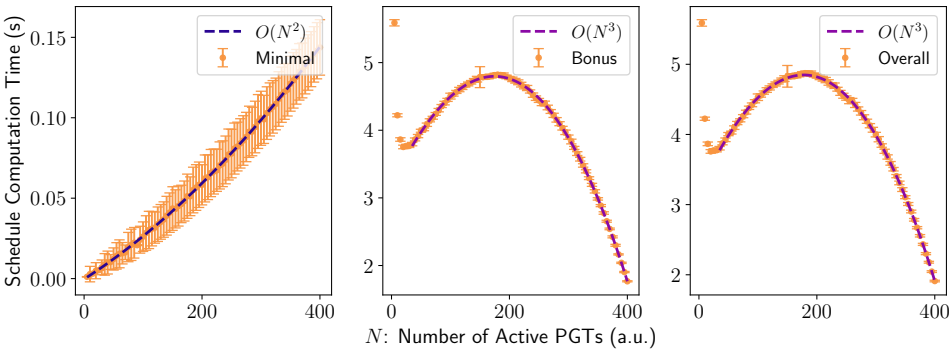


Figure 5.12: Average computation time of the minimal (left) and bonus (middle) allocation phases of Compute Schedule, and overall computation time (right), plotted against the number of active PGTs $N = |Z^*_\Phi|$. Each data point is the average of 50 repetitions of Compute Schedule for a set of active PGTs Z^*_Φ. Error bars show one standard deviation. Polynomial fits to the data are based on least squares fitting.

5

- $\mathcal{N}^{SI}_\gamma = 20,\ \forall \gamma \in Z^*_\Phi$.

To study this scenario, the scheduling interval was set to eight thousand seconds ($T^{SI} = 8000$) and we repeated schedule computation for scenarios with $|Z^*_\Phi|$ ranging from five to four hundred, in steps of five.

In Figure 5.12 the schedule computation time, decomposed into the minimal allocation phase (left), the bonus allocation phase (middle) and the overall computation time (right), is plotted against N. Data points are the average over fifty independent repetitions and error bars indicate one standard deviation. The time to complete the minimal allocation phase (left) is increasing in N and a polynomial fit validates that it has $O(N^2)$ complexity (left). The time to complete the bonus allocation phase (middle) exhibits non-trivial variation with N, yet all features are explained by our algorithmic analysis.

Whenever there is a gap in the schedule, the BONUSROUNDROBIN algorithm checks the next PGT in round robin order to determine if all the required resources are available ($O(R)$ operation). Checking γ_0 additionally requires both checking for both past and future minsep violations, since $t^{\texttt{minsep}}_{\gamma_0} \neq 0$. For small N, numerous gaps exist and the round robin index frequently revisits γ_0, resulting in the initial spike in computation time. The most time-intensive operation occurs when filling gaps between two already scheduled PGAs, which requires list insertion ($O(N)$ in our implementation). When no gaps exist but time remains in the scheduling interval, PGAs can simply be appended to the schedule ($O(1)$). For $N < 200$, gaps exists in the schedule output by the minimal allocation phase, and these are filled by list insertion. For $N \geq 200$, no gaps remain and new PGAs are only appended to the schedule by ROUNDROBINBONUS. This transition produces the local maximum at $N = 200$. As N increases further, the number of PGAs that can be appended to the schedule decreases, until $N = 400$ where the schedule output by the minimal allocation phase has no gaps and occupies the entire scheduling interval. A polynomial fit validates $O(N^3)$ complexity for the bonus allocation phase. The overall complexity (right) confirms the result of Theorem 20, showing that the schedule

computation time is dominated by the bonus allocation phase and has $O(N^3)$ complexity. See Appendix 5.10.4 for full details of our complexity analysis.

5.9. CONCLUSION AND OUTLOOK

We have designed Arqon, a novel suite of control applications for a centrally controlled quantum network that establishes and satisfies well-defined service agreements with end nodes. These service agreements address realistic application execution requirements on programmable end nodes, so that satisfying a service agreement implies that user applications can succeed. In contrast to existing quantum network control architectures, Arqon includes an admission control mechanism which prioritizes service to already accepted demands over incoming demands, preventing service disruptions.

We extended the concept of reliable service from classical computer networks and demonstrated through analysis and numeric evaluations with a proof of principle implementation that Arqon provides reliable service to all accepted demands in static network topologies. We designed Arqon to respond to changes in network topology or capabilities. In our proof of principle implementation we did not implement the mechanisms for handling these updates. A future research direction is to create concrete implementations of these mechanisms and evaluate the robustness of Arqon against changes to topology and capabilities in a variety of parameter regimes.

We developed fast algorithms for producing network schedules and executing a demand admission control process, and demonstrated numerically that each algorithm can process up to one thousand demands in less than 7 seconds. The core of these algorithms relies on directly calculating how a minimal allocation of PGAs for all active PGTs can be scheduled in each scheduling interval. In contrast, many quantum network architectures implement scheduling algorithms which rely on optimization, which is generally slower than direct calculation of a schedule.

Arqon's admission control process, which ensures arriving demands do not disrupt service to accepted demands, comes at the cost of rejecting some demands. In simulations based on our proof of principle implementation we observed that there is scope to increase the proportion of demands that Arqon accepts. We proposed modifying the calculation which Arqon's admission control process uses to estimate load on the network. To further increase the resource utilization and proportion of demands that may be accepted, we proposed defining a path partition that differentiates between paths through the network that cross one or multiple long-distance backbones.

The performance of any quantum network architecture needs to be tested in a real world deployment. We have developed an implementation of Arqon in Python as a simulator. The next step is to create a real-time implementation of Arqon, integrate it with an SDN-controller, and use the integrated system to produce network schedules for the components of a test-bed quantum network. Such a demonstration would bring scalable quantum networking one step closer to reality.

5.10. APPENDICES

5.10.1. NOTATION

The notation used throughout this paper is summarized in Table 5.4.

Symbol	*Definition*
	Applications
$\mathcal{S}$	An application session
$\mathcal{N}$	A set of end nodes which hosts an application session
N^{inst}	A number of application instances required to complete an application session
$t^{\texttt{expiry}}$	The expiry time of an application session and demand
	Demands
$\mathfrak{p} = (w, s, F)$	A packet of s end-to-end links to be generated within time window w, with minimum fidelity F
$t^{\texttt{minsep}}$	A minimum required time separation between attempts to generate a packet
$d = (d^{\text{raw}}, M_d)$	The full demand consists of the raw demand d^{raw} and demand metadata M_d
$\epsilon_d^{\text{service}}$	The service error parameter for a demand d
	Packet Generation Tasks
γ	A PGT
E_γ	The duration of PGAs for the PGT γ
$p_\gamma^{\texttt{packet}}$	The probability a PGA of duration E_γ successfully produces the required packet of links
$t_\gamma^{\texttt{start}}$	The earliest time that a PGA for a PGT γ could be scheduled if it is accepted
$\delta(\gamma, t; r)$	A PGA for a PGT γ scheduled at time t on resource r
	Network Topology and Components
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	A network resource graph with vertices $\mathcal{V}$ and edges $\mathcal{E}$
$E \subset \mathcal{V}$	The set of end nodes
$I \subset \mathcal{V}$	The set of entanglement generation interfaces
$J \subset \mathcal{V}$	The set of junction nodes
$B \subset \mathcal{V}$	The set of long-distance backbones
$\mathfrak{R} = I \sqcup J \sqcup B$	Set of all internal resources in a network
$R = \lvert\mathfrak{R}\rvert$	Number of internal resources in a network
$\mathcal{L}$	The set of local areas of $\mathcal{G}$
$\mathcal{E}^* \subset \mathcal{E}$	A set of allowed edges of a network resource graph $\mathcal{G}$
	Paths
$\pi = (\pi_0, \cdots, \pi_k)$	a path through the network resource graph from source node π_0 to destination node π_k
$\mathcal{P}_{\text{valid}}$	The set of valid entanglement generation paths
$\mathcal{P}_{\text{allowed}} \subset \mathcal{P}_{\text{valid}}$	A set of allowed paths, further restricted from the set of valid paths
$\Pi = \{\Pi_\phi\}$	A disjoint partition of $\mathcal{P}_{\text{valid}}$ or $\mathcal{P}_{\text{allowed}}$
	Demand Manager
τ	The demand termination buffer
$\mathbf{\Gamma}$	The PGT intake buffer
$\mathbf{\Gamma}_d \subseteq \mathbf{\Gamma}$	A set of PGTs in the task intake buffer which can realize a demand d
$k = \lvert\mathbf{\Gamma}\rvert$	Number of PGTs in the task intake buffer
	Network Scheduler
S	A network schedule
T^{SI}	The duration of a scheduling interval
t_{compute}	The computation time of the Network Schedule application
Φ	The set of filing classes
$\phi = (Z_\phi, \Pi_\phi)$	A filling class based on the cell of the path partition Π_ϕ so that $\pi_\gamma \in \Pi_\phi \ \forall \gamma \in Z_\phi$
Z_Φ	The set of all active PGTs
$N = \lvert Z_\Phi \rvert$	Number of active PGTs
ξ	Mapping from a set of paths to the associated internal resources

Table 5.4: Summary of the notation used throughout this paper. The notation $\boldsymbol{f}[i]$ indicates the i^{th} entry of a vector, tuple, matrix or dictionary-like object $\boldsymbol{f}$.

5.10.2. NETWORK SCHEDULER ALGORITHMS

In this appendix we provide pseudocode algorithms for all processes of the Network Scheduler for which an algorithm is specified in the main text. These are as follows:

- UPDATEFILLINGCLASSES, Algorithm 9 for Update Filling Classes.
- ADMITTASKS, Algorithm 10 for Admit Tasks.
- DIRECTALLOCATION, Algorithm 11, which implements the inner loop of the minimal allocation phase of Compute Schedule.
- BONUSROUNDROBIN, Algorithm 12, which implements the inner loop of the bonus allocation phase of Compute Schedule.
- NEXTSTARTTIME, ALLRESOURCESAVAILABLE and GETLEFT[RIGHT]MINSEPVIOLATIONS are all in Algorithm 13. These functions are used by the BONUSROUNDROBIN algorithm to determine whether additional PGAs can be added to a network schedule.

Algorithm 9: Algorithm for Update Filling Classes. The functions BUILDFILLINGCLASSES, ASSIGNFILLINGCLASS, and APPLYRULEMISSINGPATH may be re-defined by an implementation, or they may follow the default implementation of Section 5.6.

Function UPDATEFILLINGCLASSES**:**
  Input : Set of filling classes $\Psi = \{Z_\psi, \Pi_\psi\}_\psi$, Path partition Π, Set of terminated PGTs τ.
  Output : Updated set of filling classes $\Phi = \{Z_\phi, \Pi_\phi\}_\phi$, Mapping of associated resources ξ.
  Set $\Gamma^* \leftarrow \emptyset$;
  Set $\Phi, \xi \leftarrow$ BUILDFILLINGCLASSES$(Z = \emptyset, \Pi)$;
  for $\psi \in \Psi$**:**
    for $\gamma \in Z_\psi$**:**
      if $\gamma \in \tau$ **or** $t_\gamma^{\texttt{expiry}} \leq$ `CurrentTime`**:**
        continue;
      if $\exists \phi$ *s.t.* $\pi_\gamma \in \Pi_\phi$ **then**
        Set $\Phi \leftarrow$ ASSIGNFILLINGCLASS(γ, Φ);
      else
        Set $\boldsymbol{\Gamma}^* \leftarrow \boldsymbol{\Gamma}^* \cup \gamma$;
      end
  for $\gamma \in \boldsymbol{\Gamma}^*$**:**
    Set $\Phi \leftarrow$ APPLYRULEMISSINGPATH(γ, Φ) ;
end
return Φ, ξ;

Algorithm 10: The Admit Tasks process of the Network Scheduler. The function GETFILLINGCLASS returns the filling class of a PGT if it exists, or else it raises a `FillingClassError`. CALCULATEREQUIREDTIME is defined in Algorithm 7.

Function ADMITTASKS**:**
  Input : Task Intake Object $\mathbf{\Gamma}$, Set of filling classes Φ, Set of resources $\mathfrak{R}$.
  Output : Set of Filling classes Φ, List of accepted PGTs $\boldsymbol{A}$.
  Set $T_{\xi_r} \leftarrow T^{SI}$ for $r \in \mathfrak{R}$;
  for $\phi \in \Phi$**:**
    $R(Z_\phi) =$ CALCULATEREQUIREDTIME(Z_ϕ) ;
    for $r \in \xi(\Pi_\phi)$**:**
      Set $T_{\xi_r} \leftarrow T_{\xi_r} - R(Z_\phi)$;
  while $\mathbf{\Gamma} \neq \emptyset$**:**
    Set $\Gamma_d \leftarrow \mathbf{\Gamma}[0]$;
    while $\Gamma_d \neq \emptyset$**:**
      Set $\gamma \leftarrow \Gamma_d[0]$;
      try:
        Set $\phi \leftarrow$ GETFILLINGCLASS(γ) ;
      except `FillingClassError`**:**
        Set $\Gamma_d \leftarrow \Gamma_d \setminus \gamma$;
        continue;
      Set $\tilde{Z}_\phi \leftarrow Z_\phi + \gamma$;
      Set `Accept` $\leftarrow$ True ;
      Set $T_{\xi_r} \leftarrow T_{\xi_r} + R(Z_\phi)$ for $r \in \xi(\Pi_\phi)$;
      Set $R(\tilde{Z}_\phi) \leftarrow$ CALCULATEREQUIREDTIME$(\tilde{Z}_\phi)$;
      for $r \in \xi(\Pi_\phi)$**:**
        if $T_{\xi_r} - R(\tilde{Z}_\phi) < 0$**:**
          Set `Accept` $\leftarrow$ False ;
          Set $\Gamma_d \leftarrow \Gamma_d \setminus \gamma$;
          if $\Gamma_d = \emptyset$**:**
            Set $\mathbf{\Gamma} \leftarrow \mathbf{\Gamma} \setminus \Gamma_d$;
          break;
      if `Accept`**:**
        Set $\boldsymbol{A} \leftarrow \boldsymbol{A} + \gamma$;
        Set $Z_\phi \leftarrow \tilde{Z}_\phi$;
        Set $R(Z_\phi) \leftarrow R(\tilde{Z}_\phi)$;
        Set $T_{\xi_r} \leftarrow T_{\xi_r} - R(Z_\phi)$ for $r \in \xi(\Pi_\phi)$;
        Set $\mathbf{\Gamma} \leftarrow \mathbf{\Gamma} \setminus \Gamma_d$;
        break;
  return Φ, $\boldsymbol{A}$
end

Algorithm 11: Scheduler for direct allocation of PGA start times for all PGTs in a single filling class ϕ. $\mathcal{N}_x^{SI}$ is the number of PGAs, each with duration E_x, of PGT γ_x that need to be scheduled.

Function DIRECTALLOCATION**:**
  Input : Filling class $\phi = \left(Z_\phi, \Pi_\phi\right)$ with $Z_\phi = (\gamma_0, ..., \gamma_{M-1})$, network schedule S, start time t_0.
  Output : Updated network schedule S.
  Order PGTs $\gamma \in Z_\phi$ by a set of indices $X = \{1, \cdots, |Z_\phi|\}$ such that $\mathcal{N}_x^{SI} \leq \mathcal{N}_{x+1}^{SI}$
  Set $M \leftarrow |Z_\phi|$
  Set $c_x \leftarrow \max\left(\max_{y \in \{x, \cdots M\}} \left(E_y + t_y^{\texttt{minsep}}\right), \sum_{y=x}^{M} E_y\right)$;
  Set $n_x \leftarrow \mathcal{N}_x^{SI} - \mathcal{N}_{x-1}^{SI}$ for $x \geq 1$;
  Set $n_0 \leftarrow \mathcal{N}_0^{SI} - 1$;
  Set $t_{\text{start}} \leftarrow t_0$;
  for $m = 0, ..., M-1$**:**
    for $k = 0, ..., n_m - 1$**:**
      for $x = m, ..., M-1$**:**
        Set $t_{\text{offset}} \leftarrow \sum_{y=m}^{x} E_y$;
        for $r \in \pi_{\gamma_x}$**:**
          Add PGA $\delta(\gamma_x, t_{\text{start}} + k c_m + t_{\text{offset}}; r)$ to $S[r]$;
    for $r \in \pi_{\gamma_x}$**:**
      Add PGA $\delta(\gamma_m, t_{\text{start}} + n_m c_m; r)$ to $S[r]$;
    Set $t_{\text{start}} \leftarrow t_{\text{start}} + n_m c_m + E_m$;
end

Algorithm 12: Algorithm for the inner loop of the bonus allocation phase of Compute Schedule. $\mathbb{Z}_{|\phi|,k} = \{k, k+1, ..., |\phi|-1, 0, 1, ..., k-1)\}$. $\delta(\gamma, t; r)$ is a PGA for a PGT γ scheduled at time t on resource r. The subroutines NEXTSTARTTIME, ALLRESOURCESAVAILABLE and GETLEFT[RIGHT]MINSEPVIOLATIONS are specified in Algorithm 13.

Function ROUNDROBINBONUS**:**
  Input : Filling class ϕ, network schedule S.
  Output : Updated network schedule S.
  Set $T^{\text{rel}} \leftarrow \{\}$;
  Set $t^{\texttt{est}} \leftarrow$ NEXTSTARTTIME$(S, T^{\text{rel}}, 0)$;
  Set $k \leftarrow 0$;
  while $t^{\texttt{est}} < T^{SI}$**:**
    for $i \in \mathbb{Z}_{|\phi|,k}$**:**
      Set $C_{\text{resource}} \leftarrow$ ALLRESOURCESAVAILABLE$(\gamma_i, t^{\texttt{est}}; S)$
      Set $V_L^{MS} \leftarrow$ GETLEFTMINSEPVIOLATIONS$(\gamma_i, t^{\texttt{est}}; S)$
      Set $V_R^{MS} \leftarrow$ GETRIGHTMINSEPVIOLATIONS$(\gamma_i, t^{\texttt{est}}; S)$
      Set $C_{\text{minsep}} \leftarrow V_L^{MS} \cup V_R^{MS} = \emptyset$;
      if $C_{\text{resource}} \wedge C_{\text{minsep}} \wedge t^{\texttt{est}} + E_{\gamma_i} < T^{SI}$ **then**
        for $r \in \pi_{\gamma_i}$**:**
          Add PGA $\delta(\gamma_i, t^{\texttt{est}}; r)$ to S_r;
        Set $k \leftarrow i$;
      else if $V_L^{MS} \neq \emptyset$ **then**
        Append $\max_{\delta \in V_L^{MS}}(t_{\delta^*}^{\texttt{end}} + t_{\gamma_i}^{\texttt{minsep}})$ to T^{rel}
      end
    Set $t^{\texttt{est}} \leftarrow$ NEXTSTARTTIME$(S, T^{\text{rel}}, t^{\texttt{est}})$;
end

Algorithm 13: Auxiliary algorithms used in BONUSROUNDROBIN, Algorithm 12.

Function NEXTSTARTTIME**:**
  Input : Network Schedule S, list of release times $\hat{T}^{\text{rel}}$, current time t
  Output : next possible start time $t^{\texttt{est}}$, updated set of release times T^{rel}
  Set $S' = \bigcup_{r \in \mathfrak{R}} \{\delta \in S_r : t_\delta^{\texttt{end}} > t\}$;
  if $S' = \emptyset \wedge \hat{T}^{\text{rel}} = \emptyset$**:**
    return ∞
  Set $t_s = \min\{t_\delta^{\texttt{end}} : \delta \in S'\}$;
  Set $t_r = \min \hat{T}^{\text{rel}}$;
  if $t_r \leq t_s$**:**
    Set $T^{\text{rel}} \leftarrow \hat{T}^{\text{rel}} \setminus t_r$;
  return $\min\{t_s, t_r\}, T^{\text{rel}}$;
end
Function ALLRESOURCESAVAILABLE**:**
  Input : PGT γ, time t, network schedule S
  Output : boolean
  for $r \in \pi_\gamma \cap \mathfrak{R}$**:**
    if r *is not available in* $[t, t + E_\gamma)$**:**
      return *False*
    ;
  return *True*;
end
Function GETLEFTMINSEPVIOLATIONS**:**
  Input : PGT γ, time t, network schedule S
  Output : set of conflicting PGAs, V_L^{MS}
  Set $V_L^{MS} \leftarrow \emptyset$;
  for $r \in \pi_\gamma \cap \mathfrak{R}$**:**
    Set $S' \leftarrow \{\delta \in S_r : [t_\delta^{\texttt{start}}, t_\delta^{\texttt{end}}) \cap (t - t_\gamma^{\texttt{minsep}}, t] \neq \emptyset\}$;
    Set $V_L^{MS} \leftarrow V_L^{MS} \cup \{\delta \in S' : \text{DEMANDID}(\delta) = \text{DEMANDID}(\gamma)\}$;
  return V_L^{MS};
end
Function GETRIGHTMINSEPVIOLATIONS**:**
  Input : PGT γ, time t, network schedule S
  Output : set of conflicting PGAs, V_R^{MS}
  Set $V_L^{MS} \leftarrow \emptyset$;
  for $r \in \pi_\gamma \cap \mathfrak{R}$**:**
    Set $S' \leftarrow \{\delta \in S_r : [t_\delta^{\texttt{start}}, t_\delta^{\texttt{end}}) \cap (t, t + t_\gamma^{\texttt{minsep}}) \neq \emptyset\}$;
    Set $V_R^{MS} \leftarrow V_R^{MS} \cup \{\delta \in S' : \text{DEMANDID}(\delta) = \text{DEMANDID}(\gamma)\}$;
  return V_R^{MS};
end

5.10.3. Performance Analysis

The main goal of this Appendix is to prove Theorem 14, which is a statement about the performance of the Network Scheduler control application. To prove this theorem, we first develop a series of intermediate results which support the final analysis. This section is organized as follows:

1. We define a method of comparing filling classes and a *well-behaved* property for an abstract set of filling classes. Subsequent results for the performance of the Admit Tasks and Compute Schedule processes require an abstract set of filling classes to have this property.

2. We define sequentially valid schedules, define conditions under which a schedule for a single filling class may be guaranteed to be sequentially valid, and we calculate the required duration of sequentially valid schedules for a single filling class.

3. We define valid schedules and prove that the compute schedule process produces valid schedules.

4. We prove that the admission control process is sound, meaning that newly accepted demands do not disrupt existing service agreements *(C3)*.

5. We prove Theorem 14.

Filling Classes

In this section we first prove that each PGT is only in a single filling class, as long as the path partition Π which defines Φ is disjoint. This result is used later in the proof of 3. We then build up towards defining a well-behaved property for an abstract set of filling classes, required by Theorem 14. As a pre-curser, we define a method of comparing filling classes, $\leq_\xi$. Finally in this section we prove that a well-behaved set of filling classes in an internally connected network has a unique greatest element under the ordering $\leq_\xi$. This result is used later in the proof of Theorem 16.

Lemma 6. *Let $\Pi = \{\Pi_\phi\}$ be a disjoint path partition and Φ the set of filling classes defined by Π. Let Z be a set of PGTs such that $\forall \gamma \in Z$, $\exists \phi \in \Phi$ s.t. $\pi_\gamma \in \Pi_\phi$. Then, $\{Z_\phi\}$ is a disjoint partition of Z.*

Proof. (Disjointedness) Suppose $\exists \gamma \in Z$ such that $\gamma \in Z_\phi$ and $\gamma \in Z_\psi$. This implies that $\pi_\gamma \in \Pi_\phi$ and $\pi_\gamma \in \Pi_\psi$, which implies $\Pi_\phi \cap \Pi_\psi \neq \emptyset$. This contradicts that Π is a disjoint partition and so no such γ may exist.

(Completeness) By the hypothesis, $\forall \gamma \in Z$, $\exists \phi \in \Phi$ such that $\pi_\gamma \in \Pi_\phi$. Hence every $\gamma \in Z$ is a member of some Z_ϕ. □

Definition 48 (Comparison of filling classes)**.** *We define a partial order on Φ by $\phi >_\xi \psi \Leftrightarrow \xi(\Pi_\psi) \subset \xi(\Pi_\phi)$, with equality iff $\xi(\Pi_\psi) = \xi(\Pi_\phi)$.*

As the ordering $\geq_\xi$ is only a partial order, there may exist non-comparable elements of Φ using $\geq_\xi$. In particular, if $\xi(\Pi_\psi) \cap \xi(\Pi_\phi) = \emptyset$ then ψ and ϕ do not have a determinate order under $\geq_\xi$.

Recall that the network resource graph $\mathscr{G}$, from which the set of valid paths $\mathscr{P}_{\text{valid}}$ and the path partition Π of $\mathscr{P}_{\text{valid}}$ are derived, may have the property of being internally connected. In that case we write that the path partition Π describes an internally connected network.

Definition 49 (Well-behaved set of filling classes)**.** *Let $\Phi = \{\phi_0, \phi_1, ..., \phi_{n-1}\}$ be a set of filling classes defined by the path partition Π. Let ξ be the mapping of between a set of paths and their associated resources. We say that Φ is* **well-behaved** *if the following conditions are met:*

1. $\forall i \neq j, \xi(\Pi_{\phi_i}) \neq \xi(\Pi_{\phi_j})$
2. $\forall i \neq j, \xi(\Pi_{\phi_i}) \cap \xi(\Pi_{\phi_j}) \neq \emptyset \implies \xi(\Pi_{\phi_i}) \subset \xi(\Pi_{\phi_j}) \ \vee \ \xi(\Pi_{\phi_j}) \subset \xi(\Pi_{\phi_i})$.

With the ordering $\leq_\xi$, condition 2. may be re-stated as:

2'. $\forall i \neq j, \ \xi(\Pi_{\phi_i}) \cap \xi(\Pi_{\phi_j}) \neq \emptyset \implies \phi_j >_\xi \phi_i \ \vee \ \phi_i >_\xi \phi_j$.

These conditions ensure that firstly no two filling classes can be associated with exactly the same set of resources and secondly that if there is any overlap of associated resources between filling classes then the filling classes are comparable.

Lemma 7. *Let the path partition Π describe an internally connected network. Let $\Phi = \{\phi_{n-1}, \phi_{n-2}, \cdots, \phi_1, \phi_0\}$ be a well-behaved set of filling classes defined by Π. Then under $\leq_\xi$ as in Definition 48, there is a unique greatest element of Φ.*

Proof. The existence of a path partition Π with at least one cell in the partition defines a set of filling classes with at least one filling class. In the case that the partition has only a single cell Π_0, then there is a single filling class ϕ_0, and ϕ_0 is a unique greatest element of Φ.

Suppose therefore that the path partition has $n > 1$ cells, and therefore defines a set of $n > 1$ filling classes Φ. For the sake of contradiction, suppose Φ has no unique greatest element.

Then there exists i, j such that there is no k satisfying $\phi_k >_\xi \phi_i$ and $\phi_k >_\xi \phi_j$. Such ϕ_i, ϕ_j are greatest elements of Φ. As ϕ is well-behaved, we must have that ϕ_i and ϕ_j are incomparable under $\geq_\xi$ and so $\xi(\Pi_{\phi_i}) \cap \xi(\Pi_{\phi_j}) = \emptyset$.

As Π describes an internally connected network, there exists some path π^* which passes through nodes in $\xi(\Pi_{\phi_i})$ and nodes in $\xi(\Pi_{\phi_j})$. As Π is a disjoint partition, there exists a unique k^* such that $\pi^* \in \Pi_{\phi_{k^*}}$. As π^* passes through nodes in $\xi(\Pi_{\phi_i})$, we must have that $\xi(\Pi_{\phi_{k^*}}) \cap \xi(\Pi_{\phi_i}) \neq \emptyset$. As Φ is well-behaved, this implies that either $\xi(\Pi_{\phi_i}) \subset \xi(\Pi_{\phi_{k^*}})$ or $\xi(\Pi_{\phi_{k^*}}) \subset \xi(\Pi_{\phi_i})$. If the former case holds, then $\phi_{k^*} >_\xi \phi_i$, contradicting that ϕ_i is a greatest element of Φ. If the latter case holds, then $\xi(\{\pi^*\}) \subseteq \xi(\Pi_{\phi_{k^*}}) \subset \xi(\Pi_{\phi_i})$. However, as $\xi(\{\pi^*\}) \cap \xi(\Pi_{\phi_j}) \neq \emptyset$, then $\xi(\Pi_{\phi_i}) \cap \xi(\Pi_{\phi_j}) \neq \emptyset$ contradicting that ϕ_i and ϕ_j are incomparable under $\geq_\xi$.

The same logic applies symmetrically to ϕ_j, and therefore no such i, j can exist and the result holds. □

SCHEDULES FOR A SINGLE FILLING CLASS

In the following, we first introduce a property called a *minsep violation*. We then make use of this property to define a sequentially valid schedule for a single filling class. We then prove that a scheduling method that adheres to three simple conditions produces sequentially valid schedules for a single filling class.

Definition 50 (Minsep violation)**.** *Let S be a schedule for filling class ϕ with $Z_\phi = \{\gamma_0, \cdots, \gamma_{M-1}\}$ with start times $s_{x,i}$ and end times $e_{x,i}$ for the ith scheduled PGA of PGT γ_x, where $x \in \{0, \cdots, M-1\}$. We say a* **minsep violation** *occurs in S if $\exists x, i$ such that $s_{x,i+1} - e_{x,i} < t_{\gamma_x}^{\texttt{minsep}}$.*

Definition 51 (Sequentially valid)**.** *A schedule is said to be* ***sequentially valid*** *if it has the following two properties:*

1. *No two PGAs are scheduled simultaneously, and*
2. *No minsep violations occur.*

5

We now define two simple conditions on the start and end times of PGAs in a network schedule that are compatible with sequentially valid schedules.

Definition 52 (First and Second Sequential Scheduling Conditions)**.** *Let S be a schedule for an arbitrary filling class $\phi = (Z_\phi, \Pi_\phi)$, populated by PGTs $Z_\phi = \{\gamma_x\}_x$. Let $s_{x,i}$ and $e_{x,i}$ denote the start and end times, respectively, for the ith PGA from PGT γ_x. Let E_{γ_x} denote the execution time of a PGA for PGT γ_x. The* ***sequential scheduling conditions*** *are,*
(Condition c_1):

$$e_{x,i} = s_{x,i} + E_{\gamma_x}, \ \forall x \geq 0, \tag{5.24}$$

and,
(Condition c_2):

$$s_{x,i} = e_{x-1,i}, \ \forall x \geq 1. \tag{5.25}$$

Calculating the Required Duration of Sequentially Valid Schedules

Lemma 8. *Let $\phi = (Z_\phi, \Pi_\phi)$ with $Z_\phi = \{\gamma_0, ..., \gamma_{M-1}\}$ be a filling class such that $\forall \gamma \in Z_\phi$, the minimum allocation satisfies $\mathcal{N}_\gamma^{SI} = \overline{N}$. Let S be a schedule for ϕ satisfying the sequential scheduling conditions, (5.24) and (5.25). The schedule S is sequentially valid if and only if $\forall i = 0, ..., \overline{N} - 2$:*

$$s_{0,i+1} - s_{0,i} \geq \max\left(\max_{\gamma \in Z_\phi}\left(E_\gamma + t_\gamma^{\texttt{minsep}}\right), \sum_{\gamma \in Z_\phi} E_\gamma\right). \tag{5.26}$$

Once the start time of the very first PGA of PGT γ_0 is known, conditions (5.24) and (5.25) define a method of adding the i^{th} PGA of PGTs $\gamma_x = \{\gamma_1, \cdots, \gamma_{M-1}\}$ to a schedule, but these conditions do not specify how to add the $(i+1)^{\text{th}}$ PGA of PGT γ_0 to the schedule. For that, (5.26) is required to ensure the schedule produced is sequentially valid. The very first PGA of PGT γ_0 can be added at an arbitrary start time t_0.

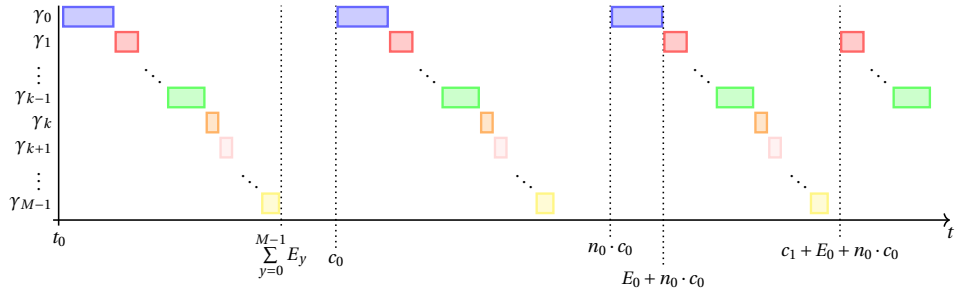


Figure 5.13: Example DIRECTALLOCATION schedule for a filling class ϕ populated by PGTs $Z_\phi = \{\gamma_0, \gamma_1, \cdots, \gamma_{M-1}\}$. In this example, $\mathcal{N}_0^{SI} = 3$, hence task γ_0 drops out after $n_0 = 2$ executions of cycle c_0. Then, one final PGA for γ_0 is scheduled. All other PGTs γ_x in the example satisfy $\mathcal{N}_x^{SI} > 4$, hence they occur in at least two executions of cycle c_1. Here also it is illustrated that in cycle c_0, task γ_0 is the task which satisfies the maximum in (5.48), hence the amount of idle time inserted in c_1 differs from that inserted in c_0.

Proof. By the hypothesis, the schedule S contains exactly $\overline{N}$ PGAs of each $\gamma \in Z_\phi$. Suppose $j \leq \overline{N} - 2$. Beginning from $s_{0,j}$, the start time of the j^{th} PGA of PGT γ_0, repeated application of conditions c_1 and c_2 generate the sequence of start and end times of the j^{th} PGAs of each PGT γ_x,

$$s_{x,j} = s_{0,j} + \sum_{y=0}^{x-1} E_{\gamma_y}, \ \forall x \geq 1, \tag{5.27}$$

$$e_{x,j} = s_{0,j} + \sum_{y=0}^{x} E_{\gamma_y}, \ \forall x \geq 0. \tag{5.28}$$

($\Rightarrow$) Suppose the schedule S has properties c_1 and c_2 of a sequentially valid schedule. By (5.24), the start time for the $(j+1)^{\text{th}}$ PGA of PGT γ_0 must satisfy $s_{0,j+1} \geq e_{M-1,j}$. By (5.28), that is

$$s_{0,j+1} \geq s_{0,j} + \sum_{y=0}^{M-1} E_{\gamma_y}. \tag{5.29}$$

By (5.25), $\forall x \geq 0$,

$$s_{x,j+1} - s_{x,j} \geq E_{\gamma_x} + t_{\gamma_x}^{\texttt{minsep}}. \tag{5.30}$$

Combining (5.27) and (5.30), $\forall x \geq 1$,

$$s_{0,j+1} + \sum_{y=0}^{x-1} E_{\gamma_x} - (s_{0,j} + \sum_{y=0}^{x-1} E_{\gamma_x}) \geq E_{\gamma_x} + t_{\gamma_x}^{\texttt{minsep}}$$

$$s_{0,j+1} - s_{0,j} \geq E_{\gamma_x} + t_{\gamma_x}^{\texttt{minsep}} \tag{5.31}$$

Combining (5.30), which holds for $x \geq 0$ and (5.31), which holds for $x \geq 1$,

$$s_{0,j+1} - s_{0,j} \geq \max_x (E_{\gamma_x} + t_{\gamma_x}^{\texttt{minsep}}). \tag{5.32}$$

The combined requirement from (5.29) and (5.32) is,

$$s_{0,j+1} - s_{0,j} \geq \max\Big(\max_x (E_{\gamma_x} + t_{\gamma_x}^{\texttt{minsep}}), \sum_{y=0}^{M-1} E_{\gamma_y}\Big).$$

($\Leftarrow$) Suppose the start times $s_{0,i},\ s_{0,i+1}\ \forall i \in \{0,\cdots,\overline{N}-2\}$ satisfy (5.26).
It follows from (5.26) and (5.28),

$$s_{0,j+1} \geq \sum_{y=0}^{M-1} E_{\gamma_x} + s_{0,j}$$
$$s_{0,j+1} \geq e_{M-1,j}. \tag{5.33}$$

Condition c_2 guarantees that the j^{th} PGAs of any PGT are non-overlapping. Then, (5.33) guarantees that the $(j+1)^{\text{th}}$ and j^{th} PGAs of any PGT are non-overlapping. Since j is arbitrary, it follows that S has property c_1.

In consideration of property c_2, it follows from (5.26) that,

$$s_{0,j+1} \geq \max_{y\geq 0}(E_{\gamma_y} + t^{\texttt{minsep}}_{\gamma_y}) + s_{0,j}. \tag{5.34}$$

Combination of (5.27) for $s_{x,j+1}$ with (5.34) gives,

$$s_{x,j+1} \geq \max_{y\geq 0}(E_{\gamma_y} + t^{\texttt{minsep}}_{\gamma_y}) + s_{0,j} + \sum_{y=0}^{x-1} E_{\gamma_y}, \tag{5.35}$$

which holds for $x \geq 1$. Subtracting (5.28) from (5.35), for $x \geq 1$ gives

$$\begin{aligned} s_{x,j+1} - e_{x,j} &\geq \max_{y\geq 0}(E_{\gamma_y} + t^{\texttt{minsep}}_{\gamma_y}) - E_{\gamma_x} \\ &\geq t^{\texttt{minsep}}_{\gamma_x}. \end{aligned} \tag{5.36}$$

Similarly, for $x = 0$, subtracting (5.28) for $e_{0,j}$ from (5.34) gives,

$$\begin{aligned} s_{0,j+1} - e_{0,j} &\geq \max_{y\geq 0}(E_{\gamma_y} + t^{\texttt{minsep}}_{\gamma_y}) - E_{\gamma_0} \\ &\geq t^{\texttt{minsep}}_{\gamma_0}. \end{aligned} \tag{5.37}$$

Since j is arbitrary, it follows that S has property c_2, no minsep violations occur in S. □

Proposition 3. *Let $\phi = (Z_\phi, \Pi_\phi)$ be a filling class. Let $R(Z_\phi) = \textsc{CalculateRequiredTime}(Z_\phi)$ denote the output of Algorithm 7. Then, $R(Z_\phi)$ is an upper bound for the minimum time required to execute a sequentially valid schedule which contains $\mathcal{N}_\gamma^{SI}$ PGAs of every PGT $\gamma \in Z_\phi$.*

Wherever it is relevant to do so, throughout the proof of Proposition 3 we refer to Figure 5.13. Note that the Figure is not required for any of the arguments in the proof. The Figure simply provides a visual illustration of the concepts introduced in the proof, easing their comprehension.

Proof. To upper bound the minimum amount of time required to execute $\mathcal{N}_\gamma^{SI}$ PGAs of every PGT $\gamma \in Z_\phi$ such that no two PGAs are scheduled simultaneously and no minsep violations occur, it is sufficient to account for the time required, R, to execute a specific schedule that meets these criteria. We first describe a method of scheduling PGAs

based on the conditions in Lemma 8 and which leverages the possibility of inserting idle time into a schedule. We then demonstrate that this method of scheduling results in a particularly simple calculation of the required time R.

Let $M = |Z_\phi|$ and let X be an ordering of Z_ϕ such that $\mathcal{N}^{SI}_{\gamma_x} \leq \mathcal{N}^{SI}_{\gamma_{x+1}},\ \forall x \in \{0,\cdots,M-1\}$. As in Lemma 8, let $s_{x,i}$ and $e_{x,i}$ identify the start and end times of the i^{th} PGA of PGT γ_x added to the schedule. When the i^{th} PGA of PGT γ_x is added to the schedule, the start and end times are specified by condition c_1 of Lemma 8,

$$e_{x,i} = s_{x,i} + E_{\gamma_x},\ \forall x \in \{0,\cdots,M-1\}. \tag{5.38}$$

Let the term *cycle 0* refer to a schedule consisting of the i^{th} PGA from each PGT $\gamma_x \in \phi$ ($i \leq \mathcal{N}^{SI}_{\gamma_0}$) according to (5.38) and the second sequential scheduling condition (5.25),

$$s_{x,i} = e_{x-1,i},\ \forall x \in \{0,\cdots,M-1\}. \tag{5.39}$$

Let cycle 0 also include an idle time of duration I_0 following the i^{th} PGA of PGT γ_{M-1}, so that

$$s_{0,i+1} = s_{0,i} + \sum_{y=0}^{M-1} E_y + I_0. \tag{5.40}$$

Note that cycle 0 has the property that no two PGAs are scheduled simultaneously. By the hypothesis, supposing that the required network resources to execute a PGA are available, it is possible to schedule a PGA as long as doing so does not introduce a minsep violation. Since cycle 0 meets the conditions that no two PGAs are scheduled simultaneously and no minsep violations occur, it is possible to commence the schedule with an iteration of cycle 0. In Figure 5.13, an iteration of cycle 0 with idle time $I_0 = 0$ is illustrated between time t_0 and the first dotted line. If $\mathcal{N}^{SI}_{\gamma_0} > 2$, then after the first iteration of cycle 0 all PGTs $\gamma_x \in \phi$ have at least 2 PGAs which remain to be scheduled. Let us consider a scheduling method where the objective is to schedule an identical repetition of cycle 0. The earliest possible start time for this repetition of cycle 0 is the earliest possibility for $s_{0,1}$. The duration of the idle time I_0 included in cycle 0 is set such that $s_{0,1}$ occurs as soon as is possible, without violating the two requirements. By Lemma 8, with $\overline{N} = \mathcal{N}^{SI}_{\gamma_0}$, to ensure the schedule meets the requirements that no two PGAs are scheduled simultaneously and no minsep violations occur, two cases can arise for $s_{0,1}$ and I_0.

Case 1: If

$$\sum_{y=0|y\neq x}^{M-1} E_{\gamma_y} \geq t^{\texttt{minsep}}_{\gamma_x}\ \forall x \in \{0,\cdots,M-1\}, \tag{5.41}$$

then,

$$\begin{aligned} s_{0,1} &= \sum_{y=0}^{M-1} E_{\gamma_y}, \\ I_0 &= 0. \end{aligned} \tag{5.42}$$

Case 2: If (5.41) doesn't hold, there exists at least one task x such that

$$\sum_{y=0|y\neq x}^{M-1} E_{\gamma_y} < t^{\texttt{minsep}}_{\gamma_x}. \tag{5.43}$$

Then,

$$s_{0,1} = \max_{y\in\{0,\cdots,M-1\}} \left(E_{\gamma_y} + t^{\texttt{minsep}}_{\gamma_y}\right), \tag{5.44}$$

$$I_0 = \max_{y\in\{0,\cdots,M-1\}} (E_{\gamma_y} + t^{\texttt{minsep}}_{\gamma_y}) - \sum_{y=0}^{M-1} E_{\gamma_y}.$$

Figure 5.13 illustrates a scenario where case 2 applies. The first dotted line in the figure indicates the time given by (5.42), and the second dotted line indicates the time given by (5.44).

Let the term *cycle m* refer to a schedule consisting of the i^{th} PGA from each PGT $\gamma_x \in \phi$ such that

$$e_{x,i} = s_{x,i} + E_{\gamma_x},\ \forall x \in \{m,\cdots,M-1\} \tag{5.45}$$

$$s_{x,i} = e_{x-1,i},\ \forall x \in \{m,\cdots,M-1\}, \tag{5.46}$$

and the i^{th} PGA of PGT γ_{M-1} is followed by idle time I_m so that

$$s_{m,i+1} = s_{m,i} + \sum_{y=m}^{M-1} E_{\gamma_y} + I_m. \tag{5.47}$$

Note that cycle m no longer includes PGAs for the PGTs $\{\gamma_0,\cdots,\gamma_m\}$.

The basis of the scheduling method is to continue scheduling cycles of PGA executions according to (5.45), (5.46) and (5.47), which preserve the ordering X. However, we only need to schedule $\mathcal{N}^{SI}_{\gamma_x}$ PGAs for PGT γ_x. Therefore, it is necessary to account for when a PGT γ_x *drops out* of a cycle, meaning that $\mathcal{N}^{SI}_{\gamma_x} - 1$ PGAs of the PGT have already been scheduled by previous cycles. After a drop out point, the PGT that is dropping out needs to be scheduled a final time. Earlier we assumed $\mathcal{N}^{SI}_{\gamma_0} > 2$. In accounting for when PGTs drop-out, we address the other possibilities, $\mathcal{N}^{SI}_{\gamma_0} = 1,2$. Due to the ordering X, it will always be the case that the PGT or PGTs dropping out at a drop out point are the first PGTs of the cycle. Figure 5.13 illustrates a scenario wherein $\mathcal{N}^{SI}_{\gamma_0} = 3$, hence it is indicated that cycle c_0 is repeated $n_0 = 2$ times, after which point γ_0 drops out, marking a transition in the schedule from cycle 0 to cycle 1. The final execution of γ_0 is scheduled before cycle 1 begins.

It follows from (5.45), (5.46), (5.47) and Lemma 8 (with $\overline{N} = \mathcal{N}^{SI}_{\gamma_m}$) that if the duration c_m of cycle m is

$$c_m = \max\left(\max_{y\in\{m,\cdots M-1\}} \left(E_{\gamma_y} + t^{\texttt{minsep}}_{\gamma_y}\right), \sum_{y=m}^{M-1} E_{\gamma_y}\right), \tag{5.48}$$

then the schedule has the properties that no two PGAs are scheduled simultaneously, and no minsep violations occur.

Exactly n_m iterations of cycle m, with duration c_m, occur before the subsequent drop out point. The first task (task zero) will drop out after

$$n_0 = \mathcal{N}^{SI}_{\gamma_0} - 1 \tag{5.49}$$

cycles. In general, n_m cycles of duration c_m will be scheduled by our scheduling method, with n_m given by,

$$n_m = \mathcal{N}^{SI}_{\gamma_m} - \mathcal{N}^{SI}_{\gamma_{m-1}}. \tag{5.50}$$

After $\sum_{y=0}^{x} n_y$ cycles, $\mathcal{N}^{SI}_x - 1$ PGAs for PGT γ_x have been added to the schedule, since

$$\begin{aligned}
\sum_{y=0}^{x} n_y &= n_0 + \sum_{y=1}^{x} n_m \\
&= \mathcal{N}^{SI}_{\gamma_0} - 1 + \sum_{y=1}^{x} (\mathcal{N}^{SI}_{\gamma_y} - \mathcal{N}^{SI}_{\gamma_{y-1}}) \\
&= \sum_{y=0}^{x} \mathcal{N}^{SI}_{\gamma_y} - \sum_{y=0}^{x-1} \mathcal{N}^{SI}_{\gamma_y} - 1 \\
&= \mathcal{N}^{SI}_{\gamma_x} - 1.
\end{aligned}$$

After it's drop out point, each task is scheduled one final time. In total these individual executions contribute an amount of time $\sum_{y=0}^{M-1} E_{\gamma_y}$ to the required time. In total, following our scheduling method, the time required to execute $\mathcal{N}^{SI}_{\gamma}$ PGAs of every task $\gamma \in Z_\phi$ is

$$R(Z_\phi) = \sum_{x=0}^{M-1} (n_x \cdot c_x + E_{\gamma_x}) \tag{5.51}$$

$$= \text{CALCULATEREQUIREDTIME}(Z_\phi). \tag{5.52}$$

□

THE COMPUTE SCHEDULE PROCESS PRODUCES VALID SCHEDULES

In this section we first extend the definition of sequentially valid schedules, which is only suitable for schedules covering a single filling class, to a definition of valid schedules for multiple filling classes. To do so, we first define the notion of a resource conflict. Then, we analyze the Compute Schedule process (Algorithm 8). This process includes a minimal allocation phase that relies on the DIRECTALLOCATION (Algorithm 11) subroutine and a bonus allocation phase that relies on the ROUNDROBINBONUS (Algorithm 12) subroutine. We prove that the schedules produced by Compute Schedule are valid and always include a minimum allocation of PGAs for every active PGT.

Definition 53 (Resource Conflict)**.** *Let Φ be a set of filling classes and Z_Φ the set of active PGTs in Φ. Let S be a schedule for the PGTs Z_Φ with start times $s_{\gamma,i}$ and end times $e_{\gamma,i}$ for the i^{th} scheduled PGA of PGT γ, $\forall \gamma \in Z_\Phi$. We say a* **resource conflict** *occurs in S if $\exists \gamma, \gamma'$, $\exists i, j$ such that*

$$\xi(\pi_\gamma) \cap \xi(\pi'_\gamma) \neq \emptyset,$$

and

$$s_{\gamma,i} \leq s_{\gamma',j} < e_{\gamma,i}.$$

Definition 54 (Valid schedule)**.** *A schedule is said to be* ***valid*** *if it has the following two properties:*

1. *No resource conflicts occur, and*

2. *No minsep violations occur.*

Note that the first condition of a valid schedule, prohibiting resource conflicts, is a weaker condition than the first condition of a sequentially valid schedule, prohibiting any two PGAs from being scheduled simultaneously. The later condition guarantees that there are no resource conflicts within the same filling class, but it would also prevent simultaneously scheduling PGAs from different filling classes, even if they do not have overlapping resource requirements.

Corollary 3. *The* DIRECTALLOCATION *scheduling algorithm (Algorithm 11) implements the specific scheduling method described in the proof of Proposition 3.*

Proof. The result follows by direct comparison of Algorithm 11 and the proof of Proposition 3. □

By Corollary 3 and Proposition 3, DIRECTALLOCATION produces a valid schedule for a filling class $\phi = (Z_\phi, \Pi_\phi)$ which requires an execution time $R(Z_\phi) = \text{CALCULATEREQUIREDTIME}(Z_\phi)$, as given by the output of Algorithm 7. We subsequently refer to this method of scheduling simply as direct allocation.

Proposition 3 makes it possible to account for the duration of a valid schedule for a single filling class. To progress towards proving Theorem 14, it is necessary to account for the duration of a valid schedule for a set of filling classes. The next results build towards the required accounting.

It is possible for a set of filling classes $\Theta = \{\theta_0, \theta_1, \cdots, \theta_{M-1}\}$ to have the property of being totally ordered, according to some ordering $\geq$. If Θ is totally ordered by an arbitrary ordering $\geq$, then $\theta_{i-1} \geq \theta_i \geq \theta_{i+1}\ \forall i \in \{0, 1, \cdots, M-1\}$. In general a set of filling classes Φ may only be partially ordered by an arbitrary ordering $\geq$, in which case there exist incomparable elements ϕ and ψ such that neither $\phi \geq \psi$, nor $\psi \geq \phi$.

Proposition 4. *Let Θ be a totally ordered set of filling classes conforming to Definition 42. Let Z_Θ denote the set of active PGTs in Θ. The minimum time required to execute a valid schedule in which there are $\mathcal{N}_\gamma^{SI}$ PGAs of every PGT $\gamma \in Z_\Theta$ is upper bounded by the quantity*

$$\tilde{\mathscr{R}}(Z_\Theta) = \sum_{\theta_l \in \Theta} R(Z_{\theta_l}), \tag{5.53}$$

where $R(Z_{\theta_l}) = \text{CALCULATEREQUIREDTIME}(Z_{\theta_l})$ is the output of Algorithm 7.

Proof. To upper bound the minimum amount of time required to execute $\mathcal{N}_\gamma^{SI}$ PGAs of every PGT $\gamma \in Z_\Theta$, we build upon the proof of Proposition 3 and calculate the time required to execute a specific schedule for Θ including $\mathcal{N}_\gamma^{SI}$ executions of each PGT $\gamma \in Z_\Theta$.

Let M be the number of filling classes in Θ. By the hypothesis, Θ is totally ordered, according to some particular ordering $\geq$, such that $\theta_0 \geq \theta_1 \geq \cdots \geq \theta_{M-1}$. The only restrictions which can disallow scheduling a PGA at a certain time are that it would introduce a resource conflict or a minsep violation. By Lemma 6, every PGT γ is associated with a single filling class. Since minsep violations may only occur between PGAs of the same

PGT, no minsep violations may occur between PGTs in different filling classes. Hence, in building a schedule for Θ it is always possible to schedule back-to-back executions of a sequentially valid schedule for filling class θ_l and a sequentially valid schedule for filling class θ_{l+1}. With the given ordering, the greatest element is scheduled first, and back-to-back executions mean that the start time of the schedule for filling class θ_{l+1} is equal to the end time of the schedule for filling class θ_l. In this scheme no two PGAs are ever scheduled simultaneously, hence no resource conflicts can occur.

To construct a specific schedule for Θ, it is thus sufficient to proceed by using direct allocation scheduling to schedule $\mathcal{N}_\gamma^{SI}$ executions of all PGTs $\gamma \in Z_{\theta_0}$. By Proposition 3, this schedule has duration $R(Z_{\theta_0})$. To continue construction of the schedule for Θ, use direct allocation scheduling to construct a schedule for each filling class $\theta_l,\ l \in \{2,\cdots,M-1\}$ and append this schedule to the end of the schedule for filling class θ_{l-1}. The total time required to execute this specific schedule for Θ is

$$\tilde{\mathscr{R}}(Z_\Theta) = \sum_{l=0}^{M-1} R(Z_{\theta_l}).$$

□

The ordering $\geq_\xi$ introduced in Definition 48 serves as an example of a specific ordering which may totally order the filling classes Θ in Proposition 4. Two filling classes are incomparable under $\geq_\xi$ whenever $\xi(\phi) \cap \xi(\psi) = \emptyset$. The following theorem extends the result of Proposition 4 from a set of totally ordered filling classes to a set of well-behaved filling classes, partially ordered by $\geq_\xi$, as in Definition 48. The extension is non-trivial, and requires accounting for how the totally ordered subsets of Φ can be consistently added to a schedule without introducing resource conflicts.

A set Φ that is partially ordered by an order $\leq$ contains totally ordered subsets. If $\Theta = \{\theta_0, \theta_1, \theta_2\}$ is a totally ordered subset of Φ, then $\{\theta_0\}$, $\{\theta_1\}$, $\{\theta_2\}$, $\{\theta_0, \theta_1\}$, $\{\theta_1, \theta_2\}$, and $\{\theta_0, \theta_2\}$ are also totally ordered subsets of Φ. In this example, only Θ is a candidate for a maximally sized totally ordered subset of Φ, as all of the other totally ordered subsets listed are subsets of Θ.

Theorem 16. *(Good Accounting) Let Φ be a set of well-behaved filling classes, partially ordered by $\geq_\xi$. Let Z_Φ denote the set of active PGTs in Φ. Let $\Omega = \{\Theta_0, \Theta_1, \cdots, \Theta_n\}$ be the set of all totally ordered subsets of Φ, of which there are a number n, for some $n \in \mathbb{N},\ n \geq 0$. The minimum time required to execute a valid schedule in which there are $\mathcal{N}_\gamma^{SI}$ PGAs of every PGT $\gamma \in Z_\Phi$ is upper bounded by*

$$\mathscr{R}(Z_\Phi) = \max_{\Theta_i \in \Omega} \tilde{\mathscr{R}}(Z_{\Theta_i}), \tag{5.54}$$

where $\tilde{\mathscr{R}}(Z_{\Theta_i})$ is given by (5.53).

Proof. (Good Accounting.) Suppose that path partition Π describes an internally connected network. Let $\Omega^{\max}$ be the set of maximally sized elements of Ω,

$$\Omega^{\max} := \{\Theta_i \in \Omega \mid \forall i' \neq i,\ \Theta_i \not\subset \Theta_{i'}\}, \tag{5.55}$$

where the inclusion $\subset$ is the usual set inclusion. Whenever $\Omega \neq \emptyset$, it follows by definition that $\Omega^{\max} \neq \emptyset$. Note that $\forall \phi \in \Phi$, $\exists \omega \in \Omega^{\max}$ such that $\phi \in \omega$. Therefore, $\Omega^{\max}$ contains a total number of elements m such that $0 \leq m \leq n$. Denote the elements as $\Omega^{\max} = \{\omega_0, \omega_1, \cdots, \omega_{m-1}\}$. Consider elements $\omega, \omega' \in \Omega^{\max}$, where $\omega = \{\phi_0, \ldots, \phi_{k-1}\}$ consists of k filling classes ordered by $\succeq_\xi$, and $\omega' = \{\phi'_0, \cdots, \phi'_{k'-1}\}$ consists of k' filling classes ordered by $\succeq_\xi$. The ordering implies that ϕ_0 is the greatest element of ω under $\succeq_\xi$ and ϕ'_0 is the greatest element of ω' under $\succeq_\xi$. As a preliminary, we prove the following claim:

Claim 1: Whenever $\omega \cap \omega' \neq \emptyset$, then for some index $0 \leq j \leq \min(k, k')$, ω and ω' are identical up to this j^{th} element. That is, $\phi_0 = \phi'_0$, $\phi_1 = \phi'_1, \cdots$, $\phi_{j-1} = \phi'_{j-1}$.

We proceed recursively and by contradiction. In the base case, suppose for the sake of contradiction that $\phi_0 \neq \phi'_0$. Since $\omega \cap \omega' \neq \emptyset$, $\exists \phi$ such that $\phi \in \omega$ and $\phi \in \omega'$. Since ϕ_0, ϕ'_0 are respectively the greatest elements of ω and ω' under the ordering $\succeq_\xi$,

$$\xi(\Pi_\phi) \subset \xi(\Pi_{\phi_0}), \tag{5.56}$$

$$\xi(\Pi_\phi) \subset \xi(\Pi_{\phi'_0}). \tag{5.57}$$

Therefore, $\xi(\Pi_{\phi_0}) \cap \xi(\Pi_{\phi'_0}) \neq \emptyset$. There are three possible cases. In the first case, $\xi(\Pi_{\phi_0}) = \xi(\Pi_{\phi'_0})$, and we arrive at the contradiction that $\phi_0 = \phi'_0$, since Φ is well-behaved. In the second case,

$$\xi(\Pi_{\phi_0}) \subset \xi(\Pi_{\phi'_0}). \tag{5.58}$$

Then,

$$\omega'' = \{\phi'_0, \phi_0, \phi_1, \cdots, \phi_{k-1}\} \tag{5.59}$$

is a totally ordered subset of Φ such that $\omega \subset \omega''$, and we arrive at the contradiction that $\omega \notin \Omega^{\max}$. In the third case, $\xi(\Pi_{\phi'_0}) \subset \xi(\Pi_{\phi_0})$ and the argument follows the same logic as in the second case.

If $(\omega \setminus \phi_0) \cap (\omega' \setminus \phi'_0) = \emptyset$, then ω, ω' are only identical up to the 1^{st} element. Otherwise, we can repeat the argument from the base case with $\omega \setminus \phi_0$ and $\omega \setminus \phi'_0$ as the basis. This recursive argumentation can be repeated until we arrive at an element $j \leq \min(k, k')$ such that $(\omega \setminus \phi_{j-1}) \cap (\omega' \setminus \phi'_{j-1}) = \emptyset$.

As a remark, note that a singleton $\omega = \phi \in \Phi$ can be a maximal element of Ω as long as $\xi(\Pi_\phi) \cap \xi(\Pi_{\phi'}) = \emptyset$, $\forall \phi' \neq \phi \in \Phi$.

We now proceed to construct a schedule S for Φ, with start time t_0, by constructing schedules $\forall \omega \in \Omega^{\max}$. For every $\omega \in \Omega^{\max}$, identify α_ω, the unique greatest element of ω according to the order $\succeq_\xi$. The existence of this unique greatest element is guaranteed by Lemma 7. Let S_ω be a direct allocation schedule for α_ω with start time $s_{\alpha_\omega} = t_0$. By Corollary 3, the end time of the schedule S_ω is $e_{\alpha_\omega} = t_0 + R(\alpha_\omega)$. Define $\tilde{\omega} = \omega \setminus \alpha_\omega$, a temporary update of ω excluding the greatest element. Now, by Lemma 7, $\tilde{\omega}$ has a unique greatest element β_ω, according to the order $\succeq_\xi$. Append to schedule S_ω a direct allocation schedule for β_ω with start time $s_{\beta_\omega} = e_{\alpha_\omega}$. By Corollary 3, the end time of the schedule S_ω is $e_{\beta_\omega} = t_0 + R(\alpha_\omega) + R(\beta_\omega)$. Update $\tilde{\omega} = \omega \setminus \{\alpha_\omega, \beta_\omega\}$. Continue in this manner until

there are no remaining filling classes in $\tilde{\omega}$, i.e. until $\tilde{\omega} = \emptyset$. Then, S_ω is a schedule for ω constructed in the manner of the proof of Proposition 4 which has start time t_0 and end time $e_\omega = t_0 + \sum_{\phi \in \omega} R(\phi_\omega)$. Hence the required execution time of S_ω is $\tilde{R}(\omega)$, as given by (5.53).

Claim 1 ensures that $\forall \omega, \omega' \in \Omega^{\max}$ the schedules S_ω, $S_{\omega'}$ are consistent, meaning if $\exists \phi \in \omega \cap \omega'$, then the start time s_{ϕ_ω} assigned to ϕ in schedule S_ω is the same as the start time $s_{\phi_{\omega'}}$ assigned to ϕ in schedule $S_{\omega'}$,

$$s_{\phi_\omega} = s_{\phi_{\omega'}}. \tag{5.60}$$

It follows immediately that

$$e_{\phi_\omega} = e_{\phi_{\omega'}}, \tag{5.61}$$

since in both S_ω and $S_{\omega'}$, the schedule for ϕ is a direct allocation schedule with start times obeying (5.60).

Therefore, $\forall \omega, \omega' \in \Omega^{\max}$, the schedules S_ω, $S_{\omega'}$ are either independent, which happens when $\omega \cap \omega' = \emptyset$, or else the schedules are consistent. To construct the full schedule S, it is therefore sufficient to loop through the schedules S_ω, $\forall \omega \in \Omega^{\max}$. For every schedule S_ω, $\forall \phi \in \omega$, if S does not already include the start time s_ϕ and end time e_ϕ for ϕ, these start and end times are added to S. This method guarantess that no resource conflicts occur in S. Furthermore, since $\forall \phi \in \Phi$, $\exists \omega \in \Omega^{\max}$ such that $\phi \in \omega$, the schedule S constructed in this manner includes a direct allocation schedule $\forall \phi \in \Phi$. Hence, the schedule S includes $\mathcal{N}_\gamma^{SI}$ PGAs of every PGT $\gamma \in Z_\Phi$ such that no minsep violations occur.

Let $\omega^* \in \Omega^{\max}$ indicate a set with schedule S_{ω^*} which is a solution to

$$S_{\omega^*} = \max_{\omega \in \Omega^{\max}} \tilde{R}(Z_\omega), \tag{5.62}$$

where Z_ω denotes the set of PGTs which occupy the set of filling classes ω. Whenever $\Omega^{\max} \neq \emptyset$, there exists such an ω^*, S_{ω^*}, which need not be unique. The amount of time required to execute the schedule S is the amount of time required to execute S_{ω^*}. This concludes the proof in the case that the path partition Π describes an internally connected network.

In the case that the path partition Π describes a network that is disconnected, partition Π into it's internally connected sub-networks. The preceding proof then applies to each internally connected sub-network. Let $\{S\}$ denote the set of schedules produced for the set of internally connected sub-networks. The schedules $S \in \{S\}$ are independent and have no resource conflicts since the schedules are for sub-networks of the path partition, which have no resources in common. Each schedule $S \in \{S\}$ has start time t_0 and some end time $e_S \geq t_0$. Therefore, the set of schedules $S \in \{S\}$ constructs a schedule $\mathfrak{S}$ for Φ in which $\mathcal{N}_\gamma^{SI}$ PGAs of every PGT $\gamma \in \Phi$ are scheduled, no minsep conflicts occur, and no resource conflicts occur. Moreover, $\exists S^*$ with end time e_{S^*} which satisfies $e_{S^*} = \max_{S \in \{S\}} e_S$. The schedule $\mathfrak{S}$ has required execution time $e_{S^*} = \mathscr{R}(\Phi)$, given by (5.54).

□

Corollary 4. *The* MINIMALALLOCATION *function (Algorithm 8) implements the specific scheduling method described in the proof of Theorem 16 (Good Accounting).*

Proof. The result follows by direct comparison of Algorithm 8 and the proof of Theorem 16. In particular, note that the MINIMALALLOCATION function in COMPUTESCHEDULE makes use of the partial ordering of the filling classes in building the schedule S with start time t_0. To construct S, $\forall \phi \in \Phi$, a direct allocation schedule for ϕ is inserted into S with start time s_ϕ and end time $e_\phi = s_\phi + R(\phi)$. The start time s_ϕ is set according to

$$s_\phi = t_0 + \sum_{\phi' >_\xi \phi} R(\phi'). \tag{5.63}$$

The identification of $\phi' >_\xi \phi$, implicitly describes that MINIMALALLOCATION identifies the subset $\omega \subset \Omega$ of all totally ordered subsets of Φ for which $\omega \cap \phi \neq 0$. By (5.63), $\forall \phi' \geq_\xi \phi$, $s_{\phi'} < s_\phi$ and moreover

$$s_\phi = \max_{\phi' >_\xi \phi} e_{\phi'}, \tag{5.64}$$

which is exactly how s_ϕ is assigned in the proof of Theorem 16. □

As a consequence of Theorem 16, given the set of filling classes Φ as input, the minimal allocation phase of Compute Schedule (Algorithm 8) produces a valid schedule which includes $\mathcal{N}_\gamma^{SI}$ PGAs of every PGT $\gamma \in \Phi$. The schedule requires execution time $\mathcal{R}(\Phi)$, as given by (5.53). We subsequently refer to this method of scheduling simply as minimal allocation scheduling.

Admit Tasks is Sound

For the Admit Tasks process to be sound, it must guarantee that newly accepted demands do not disrupt existing service agreements *(C3)*. Before proceeding, it is useful to introduce definitions describing the availability of resources and how much time has already been reserved on a network resource.

Definition 55 (Resource Availability)**.** *A resource is* available *between times t_1 and t_2 if there are no scheduled PGAs for any PGTs at any time between t_1 and t_2. The total amount of* **time available** *on each resource $r \in \mathfrak{R}$ is a non-negative number*

$$T_{\xi_r} \in [0, T^{SI}]. \tag{5.65}$$

Definition 56 (Time Reserved on a Resource)**.** *Let $r \in \mathfrak{R}$ and let T_{ξ_r} be the total amount of time available on r. A duration of time T is* **reserved on resource** *r by decrementing the available time by T,*

$$T_{\xi_r} \leftarrow T_{\xi_r} - T. \tag{5.66}$$

We say it is possible *to reserve time T on resource r if* (5.66) *is positive. A reservation of time with duration T may be* **canceled** *by incrementing the available time on resource r by T,*

$$T_{\xi_r} \leftarrow T_{\xi_r} + T.$$

Note that in the proofs of Propositions 3 and 4 it was assumed that the required network resources to execute a PGT are available. It remains to be shown that Algorithm 10 guarantees that a PGT is only accepted if it is possible to reserve sufficient time on the required network resources.

Theorem 17 (Admit Tasks is Sound)**.** *Let $\gamma' \in \mathbf{\Gamma}$, let Φ be a well-behaved set of filling classes with the set of active PGTs Z_Φ, and suppose $\exists \phi \in \Phi$ such that $\pi_{\gamma'} \in \Pi_\phi$. Then, γ' is accepted by Admit Tasks as implemented by Algorithm 10 if the time required to execute a minimal allocation schedule for $Z_\Phi + \gamma'$ is less than or equal to the scheduling interval. That is,*

$$\mathscr{R}(Z_\Phi + \gamma') \leq T^{SI}, \tag{5.67}$$

where $\mathscr{R}(\cdot)$ is as in (5.54).

Proof. Throughout, let $\mathscr{R}(\Theta)$ denote the time required to execute a minimal allocation schedule for a set of filling classes Θ, as given by (5.54), and let $R(\theta)$ denote the time required to execute a direct allocation schedule for a single filling class θ, as given by (5.52).

The set of active PGTs Z_Φ satisfies $\mathscr{R}(Z_\Phi) \leq T^{SI}$, as every active PGT was previously accepted by Admit Tasks. In the special case where there are no active PGTs, i.e. $Z_\Phi = \emptyset$ and $N = |Z_\Phi|$ is zero, this still holds as

$$\mathscr{R}(Z_\Phi = \emptyset) = 0 \Rightarrow \mathscr{R}(Z_\Phi) \leq T^{SI}.$$

In that case also

$$Z_\phi = \emptyset \wedge R(Z_\phi) = 0 \; \forall \phi \in \Phi.$$

This situation must occur at least the very first time Admit Tasks executes, and may occur in some scheduling interval k if all active PGTs expire or are terminated by Update Filling Classes, which precedes Admit Tasks.

In the first step of ADMITTASKS, the available time T_{ξ_r} on each resource $r \in \mathfrak{R}$ is reset to the duration of the scheduling interval, T^{SI}. For each filling class $\phi \in \Phi$, the amount of time $R(Z_\phi)$ required to execute a direct allocation schedule is calculated. Then, $\forall r \in \xi(\Pi_\phi)$ the time reserved on r is incremented by $R(Z_\phi)$. By Corollary 4, sequentially reserving the time required to execute a direct allocation schedule on a resource $r \in \xi(\Pi_\phi)$, $\forall \phi$, is equivalent to reserving time on the the resource according to a minimal allocation schedule.

Let M^{Z_Φ} denote the maximum amount of time reserved on any resource for the set of active PGTs Z_Φ,

$$M^{Z_\Phi} := \max_{r \in \mathfrak{R}} (T^{SI} - T_{\xi_r}).$$

By Corollary 4 this is equal to the duration of the minimal allocation schedule for Z_Φ. Overall,

$$M^{Z_\Phi} = \mathscr{R}(Z_\Phi) \leq T^{SI}.$$

At this point, the task γ' is considered by ADMITTASKS. By the hypothesis, $\exists \phi \in \Phi$ such that $\pi_{\gamma'} \in \Pi_\phi$, so GETFILLINGCLASS returns the filling class ϕ. The next step of ADMITTASKS is to add the PGT γ' to $\tilde{Z}_\phi = Z_\phi + \gamma'$, a temporary update of the set of active PGTs in filling class ϕ. The `Accept` variable is initialized as True. The time reservation of amount $R(Z_\phi)$ is then canceled on every resource $r \in \xi(\Pi_\phi)$. Next, the time $R(\tilde{Z}_\phi)$ required to execute a direct allocation schedule for $\tilde{Z}_\phi$ is calculated.

By the proof of Proposition 3 and Corollary 3, the direct allocation schedule for ϕ may include idle time based on the values of $t_\gamma^{\texttt{minsep}}$, $\forall \gamma \in Z_\phi$. For this reason, it is possible that the time required to execute a direct allocation schedule of $\tilde{Z}_\phi$, $R(\tilde{Z}_\phi)$ is not substantially

greater than $R(Z_\phi)$, and it may be that $R(\tilde{Z}_\phi) = R(Z_\phi)$. Canceling the time reservation of $R(Z_\phi)$ allows for correct comparison of $R(\tilde{Z}_\phi)$ to the available time on all resources $r \in \xi(\Pi_\phi)$. Overall, $R(Z_\phi) \leq R(\tilde{Z}_\phi)$.

For every resource $r \in \xi(\Pi_\phi)$, the time T_{ξ_r} available on resource r is compared to the required time $R(\tilde{Z}_\phi)$. If $\exists r \in \xi(\Pi_\phi)$ such that

$$T_{\xi_r} - R(\tilde{Z}_\phi) < 0, \tag{5.68}$$

then the `Accept` variable is set to false, the PGT γ is removed from $\mathbf{\Gamma}$, and γ' receives a reject decision. Otherwise, the value of `Accept` remains true, the set of active PGTs Z_ϕ is updated to $\tilde{Z}_\phi$, and the available time on each associated resource $r \in \xi(\tilde{Z}_\phi)$ is decremented by the required time for the filling class, $R(\tilde{Z}_\phi)$. Define $M^{Z_\Phi + \gamma'}$ and r^* respectively as the maximum amount of time reserved on any resource due to the PGTs in $Z_\Phi + \gamma'$ and the resource on which the maximum reservation occurs. By Corollary 16,

$$M^{Z_\Phi + \gamma'} = \mathscr{R}(Z_\Phi + \gamma'). \tag{5.69}$$

Since we condition on all comparisons (5.68) passing,

$$T_{\xi_r} - R(\tilde{Z}_\phi) \geq 0, \ \forall r \in \xi(\Pi_\phi). \tag{5.70}$$

Hence it is guaranteed that

$$T_{\xi_{r^*}} = T^{SI} - M^{Z_\Phi + \gamma'} \geq 0. \tag{5.71}$$

Combining (5.69) and (5.71),

$$\mathscr{R}(Z_\Phi + \gamma') \leq T^{SI}. \tag{5.72}$$

□

Theorem 17 (Admit Tasks is Sound) conditioned on inclusion of the path $\pi_{\gamma'} \in \Pi_\phi$ for some $\phi \in \Phi$. The following Lemma addresses PGTs $\gamma' \in \mathbf{\Gamma}$ for which $\nexists\, \phi$ such that $\pi_{\gamma'} \in \Pi_\phi$. This can happen in a network with a dynamically evolving network resource graph. For example, if a resource on the path π_γ becomes unresponsive between the time at which the demand was registered and the time at which $\mathbf{\Gamma}$ is retreived by the Network Scheduler.

Lemma 9. *Suppose $\gamma' \in \mathbf{\Gamma}$ and $\nexists\, \phi$ such that $\pi_{\gamma'} \in \Pi_\phi$. Then, γ' is rejected by* AdmitTasks.

Proof. Let d be the demand associated with γ'. Let $\mathbf{\Gamma}_d \subseteq \mathbf{\Gamma}$ denote set PGTs in $\mathbf{\Gamma}$ associated with the demand d. The AdmitTasks function calls the GetFillingClass function for the PGT γ'. If $\nexists\, \phi$ such that $\pi_{\gamma'} \in \Pi_\phi$, then GetFillingClass raises a `FillingClassError`, the PGT γ' is removed from $\mathbf{\Gamma}_d$, and the PGT γ' is rejected by AdmitTasks. □

Remark 4. *When a PGT γ' which realizes a demand d is accepted, the final step of* AdmitTasks *is to remove the set of alternative PGTs $\mathbf{\Gamma}_d \subseteq \mathbf{\Gamma}$ associated with the same demand. Hence $\mathbf{\Gamma}$ is updated to $\mathbf{\Gamma} \setminus \mathbf{\Gamma}_d$. This ensures that at most one PGT is admitted per demand.*

MINIMAL ALLOCATION IS GUARANTEED

It is now possible to prove one of our main results, Theorem 14 (Minimal Allocation is Guaranteed). This theorem underpins the proof of Theorem 13 (Deterministic Satisfaction of Service Agreements) in Section 5.5.

The following theorem assumes that the set of filling classes is well-behaved. In our implementation of Arqon (Section 5.6) we specify a path partition and the set of filling classes it induces. Then, we prove that this specific set of filling classes is well-behaved. This is sufficient to demonstrate the existence of well-behaved sets of filling classes.

Theorem 14 (Minimal Allocation is guaranteed)**.** *Let d be a demand accepted by Arqon as the PGT γ_d. Suppose $\pi_{\gamma_d} \in \Pi$ in every relevant scheduling interval. Suppose Φ is a well-behaved set of filling classes. Then, in every relevant scheduling interval the schedule $S = \text{COMPUTESCHEDULE}(\Phi)$ contains at least $\mathcal{N}_{\gamma_d}^{SI}$ PGAs of the PGT γ_d. Moreover, the duration of S does not exceed T^{SI}.*

Proof. By Theorem 17, the time required to execute a minimum allocation schedule does not exceed the scheduling interval T^{SI}.

It remains to show that given a schedule S_k as input, the bonus allocation phase of Compute Schedule cannot extend the required execution time of the schedule S_k to beyond T^{SI}. The BONUSALLOCATION function in Algorithm 8 for Compute Schedule takes the schedule S_k as input, and $\forall \phi \in \Phi$, it returns an update of the schedule, $S_k = \text{ROUNDROBINBONUS}(\phi, S_k)$. ROUNDROBINBONUS, given by Algorithm 12 may insert PGAs of a PGT γ into the schedule S_k. Before any PGA may be added, the following condition is checked:

$$t^{\texttt{est}} + E_\gamma < T^{SI}. \tag{5.73}$$

Condition (5.73) ensures that a PGA is only added to schedule S_k if the time required to execute the updated schedule S_k does not exceed T^{SI}. Hence, after all updates are made to S_k in the bonus allocation phase, the time required to execute S_k does not exceed T^{SI}. □

5.10.4. COMPLEXITY ANALYSIS OF THE NETWORK SCHEDULER

We now develop the complexity analysis of the Network Scheduler, from which we derive and prove Theorem 15 in Section 5.7. We calculate the complexity in terms of the number of active PGTs $N = |Z_\Phi|$ in the set of filling classes Φ, the number of active PGTs $N_\phi = |Z_\phi|$ in a single filling class ϕ, the number of PGTs $k = |\mathbf{\Gamma}|$ in the task intake object, and the number of internal resources $R = |\mathfrak{R}|$.

To bound the complexity of a process, it must eventually halt on any input. Unlike the other processed of the Network Scheduler, which are based on loops with a finite number of iterations, the bonus allocation phase is based on ROUNDROBINBONUS, which contains a `while` loop. To bound the operational complexity of ROUNDROBINBONUS, it is necessary to demonstrate that this algorithm halts on any input.

In this Appendix we first prove that ROUNDROBINBONUS halts on any input, and then we bound the complexity of each process of NETWORKSCHEDULER.

ROUND ROBIN BONUS HALTS

To prove that ROUNDROBINBONUS, Algorithm 12, halts on any input we need to show that in finite time the earliest start time $t^{\texttt{est}}$ is updated by ROUNDROBINBONUS to a value that exceeds the scheduling interval, $t^{\texttt{est}} > T^{SI}$.

Let us simplify the representation of the algorithm in the following manner:

Algorithm 14: Simplified representation of ROUNDROBINBONUS.

```
1 Function ROUNDROBINBONUS:
    Input  : Filling class φ, network schedule S^input.
    Output : Updated network schedule S.
2   Set T^rel ← {};
3   Set t^est ← 0;
4   Set k ← 0;
5   while t^est < T^SI do
6     S, T̂^rel ← f(S, T^rel, t^est);
7     Set t^est, T^rel ← NEXTSTARTTIME(S, T̂^rel, t^est);
8   end
9 end
```

In Algorithm 14 the function f represents the action of the for loop in Algorithm 12.

To prove that the algorithm halts, we need the sequences S_i, $\hat{T}_i^{\text{rel}}$, T_i^{rel} and $t_i^{\texttt{est}}$, where

$$S_{i+1}, \hat{T}_{i+1}^{\text{rel}} = f(S_i, T_i^{\text{rel}}, t_i^{\texttt{est}}), \tag{5.74}$$

$$t_{i+1}^{\texttt{est}}, T_{i+1}^{\text{rel}} = \text{NEXTSTARTTIME}(S_{i+1}, \hat{T}_{i+1}^{\text{rel}}, t_i^{\texttt{est}}), \tag{5.75}$$

and

$$t_0^{\texttt{est}} = 0, \qquad T_0^{\text{rel}} = \emptyset, \qquad \hat{T}_0^{\text{rel}} = \emptyset, \qquad S_0 = S^{\text{input}}. \tag{5.76}$$

Note that NEXTSTARTTIME removes a release time from the set of release times to consider if it is the next start time, so in general $\hat{T}_k^{\text{rel}} \neq T_k^{\text{rel}}$.

Similarly, if a PGA is scheduled, then f will add a release time to the set of release times to consider, so $T_i^{\text{rel}} \neq \hat{T}_{i+1}^{\text{rel}}$ in general.

We first show that the next earliest start time $t_k^{\texttt{est}}$ is smaller than any remaining release times to consider.

Lemma 10. $t_k^{\texttt{est}} < \min T_k^{\text{rel}}$.

Proof. Recall that $\delta = (\gamma, t; r) \in \tilde{S}$ is a PGA for PGT γ scheduled at start time t on resource r in a schedule $\tilde{S}$. The end time of such a PGA δ is $t_\delta^{\text{end}} = t + E_\gamma$. We examine the affect of NEXTSTARTTIME. There are two cases to consider. If $\min\{t_\delta^{end} : \delta \in S'_k\} < \min \hat{T}_k^{\text{rel}}$, then

$$t_k^{\texttt{est}} = \min\{t_\delta^{end} : \delta \in S'\} < \min \hat{T}_k^{\text{rel}} = \min T_k^{\text{rel}}.$$

Otherwise, if $\min \hat{T}_k^{\text{rel}} \leq \min\{t_\delta^{end} : \delta \in S'_k\}$, then

$$t^{\texttt{est}} = \min \hat{T}_k^{\text{rel}}, \text{ and } T_k^{\text{rel}} = \hat{T}_k^{\text{rel}} \setminus \{t_k^{\texttt{est}}\} \implies t_k^{\texttt{est}} < \min T_k^{\text{rel}}.$$

□

Next we show that new release times are only added when $t^{\texttt{est}}$ is the end time of a PGA in the schedule.

Lemma 11. *Suppose that for some t_k^{est}, $\nexists \delta \in S_k$ such that $t_k^{\text{est}} = t_\delta^{\text{end}}$. Then, following the application of f as in (5.74), $\hat{T}_{k+1}^{\text{rel}} = T_k^{\text{rel}}$, i.e. no release times are added by f.*

Proof. Suppose not. Then there exists some time t^* such that $t^* \in \hat{T}_{k+1}^{\text{rel}}$ but $t^* \notin T_k^{\text{rel}}$. This implies there exists a PGA δ^* from a PGT γ^* such that $t^* = t_{\delta^*}^{\texttt{end}} + t_{\gamma^*}^{\texttt{minsep}}$ and $t_{k-1}^{\texttt{est}} < t_{\delta^*}^{\texttt{end}} \leq t_k^{\texttt{est}}$.

As $t_{k-1}^{\texttt{est}} < t_{\delta^*}^{\texttt{end}}$, by the action of NEXTSTARTTIME we must have that $t_k^{\texttt{est}} \leq t_{\delta^*}^{\texttt{end}}$. This implies that $t_k^{\texttt{est}} = t_{\delta^*}^{\texttt{end}}$. But this contradicts the assumption that $t_k^{\texttt{est}}$ is not an end time of any scheduled PGA and so no such δ^*, t^* exist. □

We now show that $t_i^{\texttt{est}}$, which is updated in each loop of ROUNDROBINBONUS by NEXTSTARTTIME is strictly increasing.

Lemma 12. *For all $i > 0$, $t_{i+1}^{\text{est}} - t_i^{\text{est}} > 0$.*

Proof. We proceed by induction on the iteration index i.

For the base case $i = 0$. Consider the action of f on $S_0, T_0^{\text{rel}}, t_0^{\texttt{est}}$. The only way for a new time to be added to T^{rel} is from a past minsep violation. As there are no PGAs scheduled before time $t = 0$, there cannot be any past minsep violations at $t_0^{\texttt{est}}$. Therefore, there are no new release times added to T_0^{rel} by f and so $\hat{T}_1^{\text{rel}} = T_0^{\text{rel}} = \emptyset$.

Now consider the action of NEXTSTARTTIME on $S_1, \hat{T}_1^{\text{rel}}, t_0^{\texttt{est}}$. As $\hat{T}^{\text{rel}} = \emptyset$ and NEXTSTARTTIME only removes elements from $\hat{T}^{\text{rel}}$, we must have $T_1^{\text{rel}} = \emptyset$. Moreover, there are no release times to consider, and so the next earliest start time must come from the earliest end time of a PGA in S_1, i.e.

$$t_1^{\texttt{est}} = \min_{\delta \in S_1} t_\delta^{\texttt{end}} \geq \min_{\gamma \in Z_\Phi} E_\gamma > 0, \tag{5.77}$$

and the base case follows.

Suppose now that the result holds for all $i \leq k$. We need to consider two cases, whether $t_{k+1}^{\texttt{est}}$ is an end time of a PGA in S_{k+1} or whether it is a release time in $\hat{T}_{k+1}^{\text{rel}}$. In the first case, we have that NEXTSTARTTIME explicitly only considers PGAs with end times greater than $t_k^{\texttt{est}}$ and so we must have $t_{k+1}^{\texttt{est}} > t_k^{\texttt{est}}$. In the latter case, suppose that $\exists t^* \in \hat{T}_{k+1}^{\text{rel}}$ such that $t^* < t_k^{\texttt{est}}$, i.e. NEXTSTARTTIME would choose $t_{k+1}^{\texttt{est}} = t^*$.

There are two ways for t^* to get into $\hat{T}_{k+1}^{\text{rel}}$, either carried forward from (5.75) in iteration k or from (5.74) in iteration $k+1$. By Lemma 10 the time returned by NEXTSTARTTIME is less that the minimum value in T_k^{rel}, so it must be that t^* was added by f as in (5.74), in iteration $k+1$. In particular, $\exists \delta^* \in S_k$ such that δ^* is a left-minsep violation for some PGT γ^* at time $t_k^{\texttt{est}}$, i.e. $t^* = t_{\delta^*}^{\texttt{end}} + t_{\gamma^*}^{\texttt{minsep}} > t^{\texttt{est}}$. This contradicts that $t^* < t^{\texttt{est}}$, and so no such t^* can exist.

Therefore, $t_{k+1}^{\texttt{est}} > t_k^{\texttt{est}}$ and the result follows. □

We now show that the sequence $t_k^{\texttt{est}}$ visits the end time of every scheduled PGA in a finite number of steps. To do so, we introduce the notation

$$S_\infty := \lim_{k\to\infty} S_k. \tag{5.78}$$

As S_k is strictly increasing, in the sense that PGAs can only be added to a schedule, this limit is well-defined. Note that in the limit $k \to \infty$ we don't impose a cutoff at the end of a scheduling interval, and instead assume the functions f and NEXTSTARTTIME are applied as long as the algorithm does not halt.

Corollary 5 (Finite number of steps between end times)**.** *Let $t_k^{\text{est}} < \max_{\delta\in S_\infty} t_\delta^{\text{end}}$ be the end time of a PGA in S_∞. Then $\exists l < \infty$ such that t_{k+l}^{est} is also the end time of a PGA in S_∞.*

Proof. Consider $t_i^{\texttt{est}}$ for $i > k$. If $t_i^{\texttt{est}} \neq t_{\delta'}^{\texttt{end}}$, for some $\delta' \in S_\infty$, then $t_i^{\texttt{est}}$ must be a release time. By Lemma 11, the number of release times in the set of release times cannot increase until $t_l^{\texttt{est}} = t_{\delta'}^{\texttt{end}}$ for some $\delta' \in S_\infty$. Hence for $k+1 \le i \le l$, $|\hat{T}_i^{\text{rel}}| \le |\hat{T}_{k+1}^{\text{rel}}|$. Furthermore, $|\hat{T}_{k+1}^{\text{rel}}| < |Z_\Phi| < \infty$ as each PGT can only contribute one release time. Therefore, after at most $l = |Z_\Phi|$ steps, $t_{k+l}^{\texttt{est}} = t_{\delta'}^{\texttt{end}}$ for some $\delta' \in S_\infty$. □

Corollary 6 (Visit every end time)**.** *Let δ be a PGA in S_∞ with end time t_δ^{end}. Then there exists some $k < \infty$ such that $t_k^{\text{est}} = t_\delta^{\text{end}}$.*

Proof. As PGAs can only be scheduled at the current value of $t^{\texttt{est}}$, any new end times are in the future and will therefore not be skipped over. The number of PGAs in S_∞ is countable, therefore let us enumerate them by $\delta_0, \delta_1, \ldots$ such that $t_{\delta_i}^{\texttt{end}} \le t_{\delta_{i+1}}^{\texttt{end}}$. Then by the previous corollary we have that there exists some $l < k|Z_\Phi|$ such that $t_l^{\texttt{est}} = t_{\delta_k}^{\texttt{end}}$. □

Finally, we show that for every time $c > 0$, $t_k^{\texttt{est}}$ exceeds c in a finite number of steps.

Lemma 13. *For each filling class $\phi \in \Phi$,* BONUSROUNDROBIN(ϕ, S^{input}) *generates the sequences* (5.74) *and* (5.75) *such that $\forall c \ge 0$ there exists $k < \infty$ for which $t_k^{\text{est}} \ge c$.*

Proof. As every PGT in Z_ϕ has a finite expiry time, after which no further PGAs are scheduled, there are only a finite number of PGAs in S_∞. We consider two cases, $c < \max_{\delta\in S_\infty} t_\delta^{\texttt{end}}$ and $c \ge \max_{\delta\in S_\infty} t_\delta^{\texttt{end}}$. In the former case, there exists some end time $t_{\delta^*}^{\texttt{end}} > c$. By Corollary 6, the sequence $(t_i^{\texttt{est}})$ reaches $t_{\delta^*}^{\texttt{end}}$ in finite time and so exceeds c in finite time.

In the latter case, consider the values that $(t_i^{\texttt{est}})$ takes after reaching the last end time in S_∞. Let k be such that $t_k^{\texttt{est}} = \max_{\delta\in S_\infty} t_\delta^{\texttt{end}}$, and let $\hat{T}_{k+1}^{\text{rel}}$ be the set of release times to consider after $t_k^{\texttt{est}}$. As each PGT can only contribute at most one release time to $\hat{T}_{k+1}^{\text{rel}}$, we have $r = |\hat{T}_{k+1}^{\text{rel}}| \le |Z_\phi| < \infty$. Furthermore, we have that $|\hat{T}_{k+i+1}^{\text{rel}}| = |\hat{T}_{k+i}^{\text{rel}}| - 1$ for $i = 1, \ldots, r$. Therefore, we have that $\hat{T}_{k+r}^{\text{rel}} = \emptyset$. Moreover, at $t_{k+r}^{\texttt{est}}$ there are no future end times (as nothing can be added to schedule which ends after $\max_{\delta\in S_\infty} t_\delta^{\texttt{end}} < t_{k+r}^{\texttt{est}}$) and no release times as $\hat{T}_{k+r}^{\text{rel}} = \emptyset$. Therefore, the action of NEXTSTARTTIME is to set $t_{k+r+1}^{\texttt{est}} = \infty > c$. Therefore $(t_i^{\texttt{est}})$ hits ∞ in a finite number of time steps and hence exceeds c in a finite number of time steps.

□

We can now prove that ROUNDROBINBONUS(ϕ, S) halts on all inputs.

Theorem 18. *For all possible inputs ϕ, S,* ROUNDROBINBONUS*(ϕ, S) halts.*

Proof. Suppose not. Then the algorithm must enter an infinite loop somewhere. The `For` loop has finite number of steps, $|Z_\phi|$, so will always exit. We define a 'cycle' of the round-robin scheduler to be each time the `While` condition is checked. Note that each cycle the value of $t^{\texttt{est}}$ is incremented by NEXTSTARTTIME as per the sequences (5.74), (5.75) and by Lemma 12 $t^{\texttt{est}}$ increases with each cycle. The program halts when $t^{\texttt{est}} > T^{SI}$. By Lemma 13 this occurs in a finite number of cycles and so the program will always halt. □

COMPLEXITY ANALYSIS

We determine the complexity of NETWORKSCHEDULER as in Algorithm 6 by examining each of the subroutines: UPDATEFILLINGCLASSES, ADMITTASKS, and COMPUTESCHEDULE.

Update Filling Classes

First, we examine the complexity of UPDATEFILLINGCLASSES, Algorithm 9, as implemented in Listing 5.5.

Lemma 14. UPDATEFILLINGCLASSES *as implemented in listing 5.5 has operational complexity $O((N+R)^2)$.*

Proof. The algorithm is split into two phases: building the set of filling classes from Π and then removing terminated and expired demands from each filling class.

If the path partition is unchanged, the old set of filling classes can be brought forward and there are no required operations. Otherwise, we build a set of filling classes from the new path partition. We create one filling class per cell of the path partition, then sort them and finally create a dictionary holding the ordering. Creating a filling class is an $O(1)$ operation, as it only requires value assignment.

To calculate how many filling classes there can be, consider the specific choice of filling classes given in Definition 47. Let $|B|$ be the number of long-distance backbones, $|J|$ the number of junction nodes and $|I|$ be the number of EGIs in the network, so $|B|+|J|+|I| = R$.

There is a single backbone partition Π^B. Each set of junction nodes J_i is disjoint, and so there are at most $|J|$ junction filling classes. Each EGI contributes a filling class, so there are I EGI filling classes. Therefore there are at most $1+|J|+|I| < R$ filling classes, and so there are $O(R)$ filling classes.

Creating all of the filling classes is thus $O(R)$. Sorting the set of filling classes is $O(R\log(R))$. Computing the dictionary holding the ordering requires a double iteration of the list of filling classes, which takes $O(R^2)$ time. The implementation of these processes is given in Listing 5.7.

The remaining assignments in lines 14-16 of Listing 5.7 take $O(1)$ time, because a None value is always assigned in NETWORKSCHEDULER. These properties are never initialized in this main loop and are only initialized in the numeric complexity evaluations of Section 4.6, resulting in Figures 5.11 and 5.12. The overall complexity of constructing the set of filling classes is thus $O(R^2)$.

If the path partition changes, we attempt to re-add each PGT γ to a filling class. This requires an $O(R)$ lookup to find the correct filling class and then $O(1)$ time to add γ to the filling class. Note that re-calculating the minimum required time for the filling class is delayed until the end of this whole routine. Therefore, re-adding PGTs takes $O(NR)$ time. Subsequently, updating the set of removed tasks requires creating and assigning a list of length $O(N)$.

Therefore, building the set of filling classes takes $O(1)$ time if there is no change to the path partition and $O(R^2 + NR + N)$ time otherwise.

To remove the terminated and expired demands, each filling class $\phi \in \Phi$ is treated in turn. For a given filling class ϕ with N_ϕ active PGTs, each PGT is iterated over and if it is in the set of terminated demands it is removed, or if it has expired it is removed. Checking if a PGT is in the terminated set is an $O(1)$ operation (set membership). Checking if a PGT has expired is also an $O(1)$ operation. Therefore, removing the terminated or expired PGTs takes $O(N_\phi)$ time.

After the PGTs are removed, the required time is recalculated (CALCULATEREQUIREDTIME(ϕ), implemented by lines 52-63 in Listing 5.6), at a cost of $O(N_\phi^2)$ – see Lemma 15. Therefore, updating each filling class takes $O(N_\phi^2)$ time, for a total of $O(N^2)$ time across all filling classes.

Updating the lists of removed and expired demands are $O(N_\phi)$ or $O(N)$ ((once we sum over all filling classes) operations, so are sub-leading.

Therefore, the overall complexity is $O(R^2 + NR + N) + O(N^2) = O((N + R)^2)$.

□

Admit Tasks

We now show the complexity of Admit Tasks, Algorithm 10, as implemented by Listing 5.4. It makes use of some of the methods of a filling class, implemented in Listing 5.6. First we determine the operational complexity of CALCULATEREQUIREDTIME, which Admit Tasks uses to reserve time for the minimum allocation of PGAs of active PGTs on all resources associated with a filling class.

Lemma 15. CALCULATEREQUIREDTIME(ϕ) *for a filling class ϕ as implemented in Listing 5.6 has operational complexity $O(N_\phi^2)$.*

Proof. Calculating the total required time requires calculating the vector of cycle times $\mathbf{c}^\phi = (c_i^\phi)$ and the vector of cycle numbers $\mathbf{n}^\phi = (n_i^\phi)$ as in the proof of Proposition 3 in Appendix 5.10.3. Calculating each of the n_i^ϕ each requires a single value assignment and subtraction, an $O(1)$ operation, for a total of $O(N_\phi)$ operations.

Calculating c_k^ϕ has the following steps: We obtain the list of end times plus minseps for each PGT with index greater than or equal to k, which has complexity $O(N-k)$. Obtaining the sum of the execution times is also $O(N-k)$. Finding the maximum of this list and the sum of the executions times is an $O(log(N-k))$ operation. Combining these three steps, the total complexity for computing each of the c_k^ϕ is $O(N-k)$. The total complexity for computing the c_i^ϕ is then $\sum_{k=0}^{N_\phi-1} O(N_\phi - k) = O(N_\phi^2)$

Once we have the n_i^ϕ and the c_i^ϕ terms, we need to calculate $\mathbf{n}^\phi.\mathbf{c}^\phi$, which requires

$O(N_\phi)$ multiplications, $O(N_\phi)$ additions and one value assignment, for an overall complexity of $O(N_\phi)$.

Combining these calculations, the total complexity is

$$O(N_\phi) + O(N_\phi)^2 + O(N_\phi) = O(N_\phi^2). \tag{5.79}$$

□

Theorem 19. ADMITTASKS *as implemented in Listing 5.4, operating on a set of filling classes* Φ *as implemented in Listing 5.7, has operational complexity*

$$O(k[(N+k)^2 + R]2^R + R^2). \tag{5.80}$$

Proof. Initially ADMITTASKS calculates and sets the current available time on each resource $r \in \mathfrak{R}$. Setting the initial available time of a full scheduling interval requires $O(R)$ value assignments (each $O(1)$). In our implementation of the filling classes the total required time is stored as a property of a filling class, so we can get the total required time in $O(1)$ (amortized) time. Each filling class is associated with $O(R)$ resources, so decrementing the time available ($O(1)$) for all resources associated with a particular filling class takes $O(R)$ time. There are $O(R)$ filling classes in total, so the initialization of the routine takes $O(R^2) + O(R) = O(R^2)$ time in total.

Next ADMITTASKS determines whether to admit a particular PGT $\gamma \in \mathbf{\Gamma}$. To find the filling class of γ we check each filling class ϕ sequentially to determine if the path is in the cell Π_ϕ of the path partition. Each filling class can be obtained in $O(1)$ time (dictionary query), performing the inclusion check is $O(1)$ (set inclusion), and there are $O(R)$ filling classes, for a total of $O(R)$ operations.

Suppose we find a filling class ϕ for γ, and let $N_\phi = |Z_\phi|$. First, we need to remove the contribution from the PGTs in Z_ϕ to the reserved time on each resource associated with ϕ. Updating each of the $O(R)$ resources takes $O(1)$ time (dictionary parameter query, subtraction, value assignment) for a total complexity of $O(R)$.

We can then add γ to Z_ϕ. To do so, first we need to determine at which index in the ordering it should be inserted to preserve the ordering by $\mathcal{N}^{SI}$. In our implementation this takes $O(N_\phi)$ time as we examine each element in sequence. Once the index i is found, γ can be inserted into the list, which takes $O(N_\phi - i)$ time. Finally, adding γ to Z_ϕ means the minimum required time needs to be recomputed, which by Lemma 15 takes $O(N_\phi^2)$ time. Therefore, the total operational complexity of adding γ to Z_ϕ is $O(N_\phi) + O(N_\phi - i) + O(N_\phi^2) = O(N_\phi^2)$.

Once the perspective PGT has been added to its filling class, we need to check if $R(Z_\phi) \leq T^{SI}$, in which case the PGT can be accepted. Updating the reserved time on each resource associated with ϕ takes $O(1)$ time (dictionary query, addition, value assignment) Checking (float comparison, $O(1)$) that the available time for all $O(R)$ resources is positive takes a total of $O(R)$ time.

If this condition is met (i.e. the PGT is accepted), then updating the dictionary of accepted demands takes $O(1)$ time. Otherwise, if the PGT is rejected, then we need to undergo the same procedure as above but in reverse, which has at worst the same complexity.

Therefore, determining an accept/reject decision for a PGT γ has total complexity

$$O(R) + O(N_\phi^2) + O(R) + O(1) = O(N_\phi^2) + O(R).$$

Each demand in $\mathbf{\Gamma}$ contributes at most one PGT into Z_ϕ, so we can bound $|N_\phi|$ above by $N + k$, and the total complexity of making a determination for γ is $O((N+k)^2 + R)$

For each possible path π along which PGAs for a demand d can be realized, there is a constant finite number of entanglement generation protocols which can be employed. Therefore, the total number of possible PGTs for a given demand is $O(2^R)$. There are k demands for which a decision needs to be made. Therefore the overall computational complexity is given by $O(k[(N+k)^2 + R]2^R + R^2)$. □

Compute Schedule

We determine the complexity of Compute Schedule by first determining the complexity of it's subprocesses. Fundamentally, computing schedules requires adding PGAs to the network schedule, which is implemented as in Listing 5.8. Compute Schedule has three phases: minimal allocation, bonus allocation, and schedule compilation. The minimal allocation phase computes direct allocation schedules (DIRECTALLOCATION, Algorithm 11, Listing 5.1) for each filling class. The bonus allocation phase attempts to add PGAs to the direct allocation schedules for each filling class, according to BONUSROUNDROBIN, Algorithm 12, which is implemented in Listing 5.2. Schedule compilation (COMPILESCHEDULE) is implemented in Listing 5.8.

Lemma 16. *Let the format of the network schedule be as implemented in Listing 5.8. Then, appending a PGA to the schedule has complexity $O(R)$. Inserting a PGA anywhere else in the schedule has complexity $O(RN)$.*

Proof. The process of scheduling a PGA for a PGT γ can be decomposed into three parts. Firstly, we get the resource schedules for the individual resources on the path π_γ. The network schedule is implemented as a dictionary of resource schedules keyed by their resource identifier, so we simply loop through the path π_γ and retrieve each schedule in turn (dictionary query, $O(1)$). Therefore, the complexity of getting all the schedules to update is $O(|\pi_\gamma|) = O(R)$.

Secondly, for each resource in π_γ, we need to configure the PGA(s) to be scheduled, as in lines 53-56 of Listing 5.8. This takes $O(1)$ time.

Finally, we need to add the constructed PGA(s) to the resource schedule S_r. If a PGA can be appended to the end of the resource schedule whilst maintaining time ordering of all scheduled PGAs, adding the PGA has the same (amortized) complexity as appending to a list, which is $O(1)$. If it is not known at which index a new PGA needs to be inserted to maintain time ordering, then it can be obtained using binary search in time $O(\log|S_r|)$. As S_r is a list of PGAs, the total number of which is $O(N)$, we have that finding the index required is $O(\log N)$. Let q be the index at which the new PGA should be inserted. Then performing the insertion has complexity $O(|S_r| - q) = O(N)$.

The total complexity of parts two and three combined is then $O(1)$ when appending and $O(N)$ otherwise. Since these parts need to be repeated for each resource on the path π_γ, in total the complexity is $O(R)$ when appending and $O(RN)$ otherwise. □

Lemma 17. DIRECTALLOCATION*(ϕ), as implemented in Listing 5.1 for a filling class ϕ, operating on a network schedule as implemented in Listing 5.8, has computational complexity*

$$O(N_\phi R) + O(N_\phi^2). \tag{5.81}$$

Proof. The total number of PGAs to be scheduled from filling class ϕ is $\sum_{\gamma\in\phi} \mathcal{N}_\gamma^{SI} = O(N_\phi)$. This holds since $\forall\gamma$, $\mathcal{N}_\gamma^{SI}$ is a positive integer. We proceed through Listing 5.1 line-by-line. In initializing the routine, lines 11-15, we obtain the logger, set `cycle_start_time` and set up one pointer for each resource. These are either value assignment or dictionary query operations, which are all $O(1)$.

In lines 17-45 we construct the schedule. The PGTs in a filling class are ordered by a set of indices X such that $\mathcal{N}_x^{SI} \leq \mathcal{N}_{x+1}^{SI}$. For each γ_x in increasing order $x = \{0, 1, \cdots, |Z_\phi|\}$, we add all $\mathcal{N}_{\gamma_x}^{SI}$ PGAs for γ_x to the schedule.

In lines 19-21 we retrieve the new cycle time (dictionary query, $O(1)$) and reset the cycle counter k (value assignment, $O(1)$). This portion has total complexity $O(1)$.

The two `for` loops, in lines 23-31 add all the PGAs for the PGT γ_x to the schedule, for each cycle in turn. For each of these PGAs, the offset of the start time from the start of the cycle is calculated in line 25. This requires summing the elements of a list of length order $O(N_\phi)$, so it is an $O(N_\phi)$ operation. Then the PGA needs to be added to the schedule. By Lemma 16, this has complexity $O(R)$ as, by construction, DIRECTALLOCATION only appends PGAs to the end of resource schedules. Therefore, the total complexity incurred from adding a PGA to the schedule is $O(R) + O(N_\phi)$. Lines 33-42 simply schedule one last PGA for each PGT following the end of it's last cycle, as discussed in the proof of Proposition 3.

The total complexity is thus given by

$$\begin{aligned} &O(1) + O(N_\phi) + O(N_\phi)(O(R) + O(N_\phi)) \\ &\qquad = O(N_\phi^2) + O(N_\phi R) \end{aligned} \tag{5.82}$$

□

Lemma 18. ROUNDROBINBONUS*(ϕ) as implemented in Listing 5.2 for a filling class ϕ operating on a network schedule as implemented in Listing 5.8 has operational complexity*

$$O(N^2 N_\phi R)). \tag{5.83}$$

Proof. The overall complexity depends on the complexity of the subroutines which are detailed in Algorithm 13 and implemented in Listing 5.2.

`_get_next_start_time`: With the implementation of the network schedule in Listing 5.8, getting the end times to consider relies on getting each resource schedule $\forall r \in \mathfrak{R}$ (dictionary query, $O(1)$) and then retrieving the end time of a specified PGA (list element retrieval at known index, $O(1)$). Therefore, getting the PGA end times to consider has a total complexity of $O(R)$. The minimal end time is determined by minimizing over the set of end times to consider, an $O(R)$ operation. Therefore, the total complexity to get the next end time is $O(R) + O(R) = O(R)$. The next release time is determined by minimizing

over the list of release times supplied as an input to the subroutine. Each PGT in ϕ can contribute at most one release time to this list for a total of $O(N_\phi)$ elements. Therefore, finding the next release time is $O(N_\phi)$. The next start time is the minimum of the next end time and the next release time. Calculating it takes $O(1)$ time (comparing two values).

If the next start time is the next end time, the set of schedule pointers is updated. To update the pointers, for each resource in $\mathfrak{R}$, we check if the pointed PGA ends before or at the next start time (dictionary query, $O(1)$, and list item retrieval at known index, $O(1)$). If so, the pointer is incremented by two ($O(1)$). Therefore, the total complexity for updating the pointers is $O(R)$. Alternatively, if the next start time is the next release time, we remove the next start time from the set of release times, which is an $O(1)$ operation.

Therefore, the contributions from getting the next end time, the next release time, finding the minimum of these and then updating the release times/schedule pointers gives a total complexity of

$$O(R) + O(N_\phi) + O(1) + O(R) = O(R) + O(N_\phi). \tag{5.84}$$

`_resources_available`: To check if all the resources required to schedule a PGA from PGT γ are available, we perform the following steps: For each resource in π_γ, we check if the pointed PGT is active at time $t^{\texttt{est}}$. This consists of a dictionary query to get the resource schedule $O(1)$, list retrieval at a known index ($O(1)$) and a constant number of comparisons ($O(1)$). Therefore, checking if one resource is available at time $t^{\texttt{est}}$ is an $O(1)$ operation. The total from checking all resources is then $O(|\pi_\gamma|) = O(R)$.

`_future_minsep_violations`: To check if there are minium separation violations in the future for a PGA from PGT γ, it suffices to check only the resource schedule from the first internal resource in π_γ. Getting the relevant resource and its resource schedule is an $O(1)$ operation (list retrieval from known index, dictionary query). Initializing the offset variable is also $O(1)$. Each check which is performed is an $O(1)$ operation (comparison, property retrieval, value assignment). The number of scheduled PGAs to check is only bounded by the size of the schedule, and so there are $O(N)$ checks to perform. Combining the contributions from obtaining the resource schedule, initializing the counter and then performing the checks gives a total complexity of $O(N)$.

`_past_minsep_violations`: The process is the same as for `_future_minsep_violations`, so the complexity is $O(N)$.

We now move to the main loop of the algorithm. The maximum number of cycles is the same as the maximum number of possible start times. Each possible start time corresponds to the end of some PGA in the schedule, so there are at most

$$\sum_{\phi\in\Phi}\sum_{\gamma\in Z_\phi}\frac{T^{SI}}{E_\gamma + t_\gamma^{\texttt{minsep}}} = O(N) \tag{5.85}$$

possible start times. The expression (5.85) dictates that every PGT can contribute at most a number of PGAs based on how many times the PGA duration plus minsep fits within the scheduling interval. Thus the number of cycles is $O(N)$.

In each cycle there are N_ϕ rounds, during each of which `_resources_available`, `_future_minsep_violations`, `_past_minsep_violations` are called. The algorithm may also add a PGA to the schedule, which by Lemma 16 takes $O(RN)$ time as these PGAs

typically need to be inserted intro the schedule rather than appended to the end. At the end of each cycle, `_get_next_start_time` is called.

Overall, the complexity is

$$O(N)\Big(O(N_\phi)\big(O(R)+O(N)+O(N)+O(NR)\big)+O(R)+O(N_\phi)\Big)=O(N^2N_\phi R). \tag{5.86}$$

□

Lemma 19. *Given that the network schedule is implemented as in Listing 5.8,* COMPILESCHEDULE *has operational complexity* $O(NR)$.

Proof. The schedule is implemented as a dictionary of lists of PGAs, one per internal resource in the network, called a resource schedule. Therefore, the only operation required is to convert each PGA into its transmission format. This takes constant time for each PGA. There are $O(N)$ PGAs in each resource schedule and $O(R)$ resource schedules, and so the overall complexity for compiling the schedule is $O(NR)$. □

Theorem 20. *The program* COMPUTESCHEDULE *has operational complexity*

$$O(N^3R). \tag{5.87}$$

Proof. By Lemmas 17, 18 and 19 the total complexity is given by

$$\begin{aligned}\sum_{\phi\in\Phi}O(N_\phi R)+\sum_{\phi\in\Phi}O(N^2N_\phi R)+O(NR)&=O(NR)+O(N^3R)\\&=O(N^3R).\end{aligned} \tag{5.88}$$

□

NUMERIC EVALUATION

In Section 4.6 we discuss the simulation of randomly generated network topologies, parametrized by the number of long-distance backbones, local areas, and end nodes in the network. Table 5.5 compiles the average schedule computation times for each of these parametrizations. The time to compute the schedule is broken down by the processes Update Filling Classes, Admit Tasks, and Compute Schedule (without the Compile Schedule process). The total time to produce the network schedule includes the time for each of these processed and Compile Schedule.

5.10.5. IMPLEMENTATION: SUPPORTING RESULTS AND FORMAL DEFINITIONS

In this section we provide formal definitions for the set of allowed edges in a network resource graph and for the set of allowed entanglement generation paths that we consider in our implementation of Arqon. Then, we prove the supporting results Lemma 5 and Lemma 20 describing the properties of the specific choice of path partition and the set of filling classes it induces. Finally, we discuss our methods of setting the duration of a PGA for a given $p^{\texttt{packet}}$.

Long-distance Backbones	Local Areas	End Nodes	Time to Update Filling Classes (s)	Time to Admit PGTs (s)	Time to Compute Schedule (s) (without Compilation)	Total Time to Produce Network Schedule (s)
1	2	15	0.00011	0.00017	0.00683	0.05268
2	2	15	0.00011	0.00019	0.00707	0.05246
2	2	50	0.00025	0.00180	0.03461	0.09126
2	3	30	0.00017	0.00051	0.02276	0.07343
2	3	50	0.00027	0.00147	0.04741	0.10465
3	3	30	0.00018	0.00061	0.02406	0.07529
5	4	40	0.00031	0.00133	0.06215	0.11797
6	3	35	0.00022	0.00121	0.03738	0.09196
7	5	50	0.00059	0.00291	0.13799	0.20227
12	4	40	0.00034	0.00182	0.08051	0.14175

Table 5.5: Average computation times for each process of the Network Scheduler control application, from the simulations of randomly generated network topologies parametrized by Table 5.1. These values are based on the simulations with default Arqon, where the bonus allocation phase of the Compute Schedule process is enabled.

ALLOWED EDGES IN A NETWORK RESOURCE GRAPH

Definition 57 (Set of Allowed Edges, $\mathscr{E}^*$)**.** *Let $v_i, v_j \in \mathscr{V}$ be two vertices in the network resource graph. An edge $e = (v_i, v_j)$ is an* **allowed edge** *if the following conditions are satisfied:*

1. *If $v_i \in E$ then $v_j \in I$. Reflexively, if $v_j \in E$, then $v_i \in I$.*
2. *If $v_i \in B$, then $v_j \in J$. Reflexively, if $v_j \in B$, then $v_i \in J$.*
3. *If $v_i \in J \wedge v_j \notin B$, then $v_j \in I$. Reflexively, if $v_j \in J \wedge v_i \notin B$, then $v_i \in I$.*

The **set of allowed edges** *is*

$$\mathscr{E}^* := \{(v_i, v_j) \mid (1. \vee 2. \vee 3.) \wedge v_i,\ v_j \in \mathscr{V}\}. \tag{5.89}$$

THE SET OF ALLOWED ENTANGLEMENT GENERATION PATHS

Definition 58 (Allowed entanglement generation paths)**.** *Let $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ be an internally connected network resource graph with $\mathscr{V} = E \sqcup I \sqcup J \sqcup B$ and $\mathscr{E} \subseteq \mathscr{E}^*$. Let $\mathscr{L}$ be the set of local areas of $\mathscr{G}$. Define*

$$\pi_{e,e'} = \left\{\pi = (e, v_1, \cdots, v_k, e') \in \mathscr{P}_{\text{valid}}\right\}, \tag{5.90}$$

the set of all valid entanglement generation paths between two specific end nodes $e, e' \in E$, of variable length $k = |\pi_{e,e'}| - 2$. The set $\mathscr{P}_{\text{allowed}} \subset \mathscr{P}_{\text{valid}}$ of **allowed entanglement generation paths** *is a restriction of the set of valid entanglement generation paths, as follows*

$$\mathscr{P}_{\text{allowed}} = \left\{\pi_{e,e'}\ \forall e, e' \in E \,\middle|\, \pi_{e,e'} \in B^*(e, e')\ \wedge \forall \mathscr{L}_y \in \mathscr{L}, \pi_{e,e'} \in L_y^*(e, e', b^*)\right\}, \tag{5.91}$$

where

$$B^*(e, e') = \{\pi_{e,e'} \,\Big|\, \sum_{x=1}^{k=|\pi_{e,e'}|-2} \mathbb{1}_{(v_x \in B)} = b^{*(e,e')}\}, \tag{5.92}$$

$$L_y^*(e,e',b^*) = \{\pi_{e,e'} \in B^*(e,e') \mid \sum_{x=1}^{k=|\pi_{e,e'}|-2} \mathbb{1}_{(v_x \in \mathscr{L}_y)} \mathbb{1}_{(v_x \in I)} = l_y^*(e,e',b^*)\}, \tag{5.93}$$

and

$$b^*(e,e') = \min_{\pi_{e,e'}} \sum_{x=1}^{k=|\pi_{e,e'}|-2} \mathbb{1}_{(v_x \in B)}, \tag{5.94}$$

$$l_y^*(e,e',b^*) = \min_{\pi_{e,e'}} \sum_{x=1}^{k=|\pi_{e,e'}|-2} \mathbb{1}_{(v_x \in \mathscr{L}_y)} \mathbb{1}_{(v_x \in I)}, \tag{5.95}$$

and $\mathbb{1}_{(A)}$ is an indicator function that takes value 1 if A is true and value 0 if A is false.

Allowed paths are restricted to those that minimize the number of backbones traversed in the path between two end nodes. Of those paths, they are also shortest paths within each local area.

5

SPECIFIC CHOICE OF PATH PARTITION

Lemma 5.

$$\Pi = \{\Pi^B\} \cup \{\Pi_j\}_{j \in J} \cup \{\Pi_i\}_{i \in I}, \tag{5.15}$$

where Π^B, $\Pi_j^J \; \forall j$, and $\Pi_i^I \; \forall i$ are as in Definition 47, is a disjoint partition of $\mathscr{P}_{\text{allowed}}$.

Proof of Lemma 5. **Disjointness**: By construction, if a path is in Π^B it cannot be in any Π_j^J nor any Π_i^I, and likewise paths in any Π_j^J cannot be in any Π_i^I. Therefore, it suffices to show that any path can be in at most one Π_j^J or at most one Π_i^I.

Suppose that there exists some π, i, i' with $i \neq i'$ such that $\pi \in \Pi_i^I$ and $\pi \in \Pi_{i'}^I$. By definition of the set of allowed edges $\mathscr{E}^*$, $(i,i') \notin \mathscr{E}^*$. Since $\mathscr{E} \subseteq \mathscr{E}^*$, $(i,i') \notin \mathscr{E}$, thus $p = (u,...,i,\bullet,...,i',...,v)$ for some end nodes $u, v \in E$. The only allowed edges $(i,\bullet)$ are (i,w) for some $w \in E$, or (i,j) for some $j \in J$. In the former case, this implies $p \notin \mathscr{P}_{\text{valid}}$ and hence also $p \notin \mathscr{P}_{\text{allowed}}$. In the latter case, $\exists k \; s.t. \; j \in J_k \implies p \in \Pi_k^J$. In both cases, this contradicts that $p \in \Pi_i^I$. Therefore, no such p, i, i' can exist.

Suppose instead that there exists p, j, j' with $j \neq j'$ such that $p \in \Pi_j^J$ and $p \in \Pi_{j'}^J$. By the definitions of $\mathscr{E}^*$ and $\mathscr{P}_{\text{allowed}}$, the only paths from nodes in J_j to nodes in $J_{j'}$ is via some $b \in B$. Therefore, there exists some k such that $p_k = b$ and thus $p \in \Pi^B$. This contradicts that $p \in \Pi_i^J$ and so no such p, j, j' can exist

Completeness: We proceed by construction. Consider some path $p = (e,\bullet,...)$, for $e \in E$. By definition of $\mathscr{E}^*$, and since $\mathscr{E} \subseteq \mathscr{E}^*$, the only possible edges $(e,\bullet)$ are of the form (e,i) for some $i \in I$. Then, $p = (e,i,\bullet,\cdots)$, for some $i \in I$. By definition of $\mathscr{E}^*$, and since $\mathscr{E} \subseteq \mathscr{E}^*$, the only allowed edges are of the form (i,e') for some $e' \in E$, or (i,j) for some $j \in J$. In the former case, $p \in \Pi_i$. In the later case, since the J_k form a disjoint partition of J, $\exists k$ such that $j \in J_k$. Therefore, if $p \notin \Pi^B$, then $p \in \Pi_k^J$. Otherwise, $p \in \Pi^B$. Therefore, all paths in $\mathscr{P}_{\text{allowed}}$ are in some Π, and so Π forms a disjoint partition of $\mathscr{P}_{\text{allowed}}$.

□

SPECIFIC CHOICE OF FILLING CLASSES

Here we prove that the specific choice of filling classes used in our implementation is well-behaved.

Lemma 20 (The Specific Choice of Filling Classes is Well-Behaved)**.** *Let $\mathcal{G}$ be an internally connected network resource graph. The specific choice of filling classes Φ defined in* (5.16) *is a well-behaved set of filling classes.*

Proof. Let $\mathcal{L} = \{L_i = (\mathcal{V}_i, \mathcal{E}_i)\}$ be the set of local areas of $\mathcal{G}$. Let $J_i = J \cap \mathcal{V}_i$ be the junction nodes in sub-graph L_i. We need to show that Φ satisfies both conditions of Definition 49.

1. Distinct Resource Sets:
   Claim: $\forall \phi, \phi' \in \Phi$ with $\phi \neq \phi'$, we have $\xi(\Pi_\phi) \neq \xi(\Pi_{\phi'})$.

   Proof: We consider all possible pairs of distinct filling classes:

   (a) $\phi = \phi^B$ and $\phi' = \phi^J_j$ for some $j \in J$.

   Since every path $\pi \in \Pi^B$ contains at least one backbone node $b \in B$, we have $B \cap \xi(\Pi^B) \neq \emptyset$. By definition, $\Pi^J_j \subseteq \mathscr{P}_{\text{allowed}} \setminus \Pi^B$, so paths in Π^J_j contain no backbone nodes. Therefore $B \cap \xi(\Pi^J_j) = \emptyset$. It follows that $\xi(\Pi^B) \neq \xi(\Pi^J_j)$.

   (b) $\phi = \phi^B$ and $\phi' = \phi^I_i$ for some $i \in I$.
   Similarly to case (a), $\Pi^I_i \subseteq \mathscr{P}_{\text{allowed}} \setminus \Pi^B$, so $B \cap \xi(\Pi^I_i) = \emptyset$. But $B \cap \xi(\Pi^B) \neq \emptyset$, so $\xi(\Pi^B) \neq \xi(\Pi^I_i)$.

   (c) $\phi = \phi^J_j$ and $\phi' = \phi^J_{j'}$ for $j \neq j' \in \{1, \cdots, |\mathcal{L}|\}$.

   Paths $\pi \in \Pi^J_j$ contain at least one junction node from J_j and paths $\pi' \in \Pi^J_{j'}$ contain at least one junction node from $J_{j'}$. Since, with the exclusion of end nodes, local areas have disjoint vertex sets (by Definition 46 of local areas), $J_j \cap J_{j'} = \emptyset$. Therefore, $\xi(\Pi^J_j) \neq \xi(\Pi^J_{j'})$.

   (d) $\phi = \phi^J_j$ and $\phi' = \phi^I_i$ for some $j \in J$ and $i \in I$.

   By definition, paths in Π^I_i contain no junction nodes. Therefore, $\xi(\Pi^I_i) \cap J_{j'} = \emptyset$, $\forall j'$. Paths in Π^J_j on the other hand contain at least one junction node $j' \in J_j$, hence $\xi(\Pi^J_j) \cap J_j \neq \emptyset$. Therefore, $\xi(\Pi^J_j) \neq \xi(\Pi^I_i)$.

   (e) $\phi = \phi^I_i$ and $\phi' = \phi^I_{i'}$ for $i \neq i' \in I$.

   By definition, paths in Π^I_i must contain the entanglement generation interface i, similarly paths in $\Pi^I_{i'}$ must contain the entanglement generation interface i'. Based on Definition 57 of allowed edges, $\mathcal{E}^*$, each path in Π^I_i ($\Pi^I_{i'}$) can contain at most one entanglement generation interface. It follows that $\xi(\Pi^I_i) \neq \xi(\Pi^I_{i'})$, and moreover $\xi(\Pi^I_i) \cap \xi(\Pi^I_{i'}) = \emptyset$.

2. Intersection of Associated Resources:
   Claim: $\forall \phi, \phi' \in \Phi$ with $\phi \neq \phi'$, if $\xi(\Pi_\phi) \cap \xi(\Pi_{\phi'}) \neq \emptyset$, then either $\xi(\Pi_\phi) \subset \xi(\Pi_{\phi'})$ or $\xi(\Pi_{\phi'}) \subset \xi(\Pi_\phi)$.

   Proof: In consideration of the structure of the specific choice of path partition, we analyze each case where sets of associated resources can intersect:

(a) $\phi = \phi^B$ and $\phi' \in \{\phi^J_j : j \in J, \phi^I_i : i \in I\}$.

Consider arbitrary $r \in \xi(\Pi_{\phi'})$. Then, $\exists \pi \in \Pi_{\phi'}$ such that $r \in \pi$. By Definition 47, π is a path in $\mathscr{P}_{\text{allowed}}$ that does not contain backbone vertices. However, since $\mathscr{G}$ is internally connected (Definition 39), there exists a path $\pi_b \in \mathscr{P}_{\text{allowed}}$ that:

i. contains at least one backbone vertex (so $\pi_b \in \Pi^B$),

ii. and traverses through resource r (since $\mathscr{G}$ is internally connected and $\mathscr{P}_{\text{allowed}}$ does not decrease the network connectivity, but only restricts to a particular type of shortest path between any two nodes).

Therefore, $r \in \xi(\Pi^B)$. Since r was arbitrary, $\xi(\Pi_{\phi'}) \subseteq \xi(\Pi^B)$. Moreover, the inclusion is proper, as $B \cap \xi(\Pi_b) \neq \emptyset$ while $B \cap \xi(\Pi_{\phi'}) = \emptyset$ (from Condition 1). Hence, $\xi(\Pi_{\phi'}) \subset \xi(\Pi^B)$.

(b) $\phi = \phi^J_j$ and $\phi' = \phi^J_{j'}$ for $j \neq j' \in J$.

In case 1. (c), we noted that, with the exclusion of end nodes, local areas have disjoint vertex sets. Since paths in Π^J_j, respectively $\Pi^J_{j'}$, are entirely contained in the local areas L_j, respectively $L_{j'}$, $\xi(\Pi^J_j) \cap \xi(\Pi^J_{j'}) = \emptyset$ and this case does not satisfy the premise of condition 2.

(c) $\phi = \phi^J_j$ and $\phi' = \phi^I_i$ for some $j \in J$ and $i \in I$.

There are two sub-cases. If $i \notin \mathcal{V}_j$, then $\xi(\Pi^J_j) \cap \xi(\Pi^I_i) = \emptyset$, since distinct local areas have distinct vertex sets. Hence this sub-case does not satisfy the premise of condition 2. Alternatively, $i \in \mathcal{V}_j$ and $\xi(\Pi^J_j) \cap \xi(\Pi^I_i) \neq \emptyset$. In this sub-case, consider arbitrary $r \in \xi(\Pi^I_i)$. Then $\exists \pi_i \in \Pi^I_i$ such that $r \in \pi_i$. By definition of Π^I_i and the set of local areas, all vertices in π_i are in $\mathcal{V}_j$. By the internal connectivity of local area L_j, there exists an allowed path $\pi_j \in L_j$ that:

i. Contains at least one junction $j' \in J_j$ (so $\pi_j \in \Pi^J_j$),

ii. and traverses resource r.

Therefore, $r \in \xi(\Pi^J_j)$. Since r was arbitrary, $\xi(\Pi^I_i) \subseteq \xi(\Pi^J_j)$. Moreover, the inclusion is proper since, from 1. (d), $J_j \cap \xi(\Pi^J_j) \neq \emptyset$ while $J_j \cap \xi(\Pi^I_i) = \emptyset$. Hence, $\xi(\Pi^I_i) \subset \xi(\Pi^J_j)$.

(d) $\phi = \phi^I_i$ and $\phi' = \phi^I_{i'}$ for $i \neq i' \in I$.

In 1. (e) we showed that $\xi(\Pi^I_i) \cap \xi(\Pi^I_{i'}) = \emptyset$, and thus this case does not satisfy the premise of condition 2.

□

DETERMINING THE PGA EXECUTION TIME

We need to determine the duration of time E_d s.t.

$$\mathbb{P}\left[\text{packet generated before time } E_d\right] \geq p^{\texttt{packet}}. \tag{5.96}$$

For packets $\mathfrak{p}$ with $s = 1$, this can be done with the geometric/exponential distribution, but for packets where the number of requested pairs $s > 1$, the exact calculation of (5.96)

is not known. There are however results from *scan statistics* which give very accurate estimations of this probability for a known number of attempts to generate entanglement m and probability of each attempt succeeding. Such results can be found in, e.g. [94, 100–102]. We translate between a number of attempts m of some concrete duration and each with a probability of success p_{succ} and the rate of successful link generation R_π on an end-to-end path π for a total duration of time E_d. For full details of our treatment of this problem, see [14], Appendix C. Every PGA from a demand d is ultimately assigned an execution time E_d that is long enough that each PGA succeeds with probability $p^{\texttt{packet}}$.

5.10.6. Code Listings

Compute Schedule

Minimal Allocation

```
def minimal_allocation_scheduler(
        filling_class: FillingClass,
        network_schedule: NetworkSchedule,
        start_time: float
):
    if not filling_class:
        return

    network_schedule.add_tasks_minimal(filling_class)

    # logger = LogManager.get_resource_scheduler_logger()

    cycle_start_time = start_time

    ponters = {r: -100 for r in network_schedule.schedule.keys()}

    for i in range(len(filling_class)):

        cycle_time = filling_class.cycle_times[i]

        k = 0

        for k in range(filling_class.numbers_of_cycles[i]):
            for j, task in enumerate(filling_class[i:]):
                offset = sum(t.execution_time for t in filling_class[i:i + j])
                _start_to_schedule = time.process_time()
                network_schedule.schedule_packet_generation_attempt(
                    task, cycle_start_time + cycle_time * k + offset,
                    True,
                    pga_id=sum(filling_class.numbers_of_cycles[:i]) + k,
                    schedule_pointers=ponters)

        _start_to_schedule = time.process_time()
        network_schedule.schedule_packet_generation_attempt(
            filling_class[i],
            cycle_start_time + cycle_time * (
                filling_class.numbers_of_cycles[
                    i]), True,
            pga_id=sum(
                filling_class.numbers_of_cycles[
                    :i]) + k,
```

```
            schedule_pointers=ponters)

        cycle_start_time += filling_class[i].execution_time + (
            filling_class.numbers_of_cycles[i]) * cycle_time
```

Listing 5.1: Implementation of the minimal allocation phase of Compute Schedule, via the DIRECTALLOCATION scheduler, Algorithm 11. Note that rather than returning an updated network schedule, this subroutine simply updates the provided `NetworkSchedule` object.

Bonus Round Robin

```
def bonus_phase_round_robin_scheduler(
        filling_class: FillingClass,
        network_schedule: NetworkSchedule,
):
    # logger = LogManager.get_resource_scheduler_logger()

    if not filling_class:
        return

    _resources = [r for r in network_schedule.schedule.keys() if
                  not r & 1 << get_component_id_size() - 1]

    schedule_pointers = {r: 0 for r in _resources}

    release_times_to_consider = set()

    round_robin_index = 0

    next_start_time = 0

    network_schedule.add_tasks_bonus(filling_class)

    def _get_next_start_time() -> float | None:
        nonlocal next_start_time

        end_times = {
            r: network_schedule.schedule[r][schedule_pointers[r]].end_time for r
            in _resources if
            schedule_pointers[r] < len(network_schedule.schedule[r])}

        # logger.debug(f'considering end times: {end_times}')

        _next_end_time = min(end_times.values()) if end_times.values() else inf
        # logger.debug(f"next end time: {_next_end_time}")

        _minimising_resources = [
            r for r in end_times.keys() if
            math.isclose(end_times[r],
                         _next_end_time)] if _next_end_time != inf else []

        _index, _next_release_time = min(
            enumerate(release_times_to_consider),
            key=lambda t: t[1]
        ) if release_times_to_consider else (0, inf)

        # logger.debug(f"considering release times: {release_times_to_consider}")
        # logger.debug(f"next release time: {_next_release_time}")
```

```
48
49          if _next_end_time == inf and _next_release_time == inf:
50              return None
51
52          if _next_end_time <= _next_release_time:
53              for r in _minimising_resources:
54                  schedule_pointers[r] += 2
55
56              if _next_end_time in release_times_to_consider:
57                  release_times_to_consider.remove(_next_end_time)
58                  # logger.debug("Also removed end time from release_times_to_consider
                        ")
59              return _next_end_time
60
61          else:
62
63              release_times_to_consider.remove(_next_release_time)
64              return _next_release_time
65
66      def _task_indices_to_inspect(start: int) -> Generator[int, None, None]:
67          for i in range(len(filling_class)):
68              yield (start + i) % len(filling_class)
69
70      def _resources_available(_tts: PacketGenerationTask) -> bool:
71          for resource in _tts.path[1:-1]:
72              if schedule_pointers[resource] >= len(
73                      network_schedule.schedule[resource]):
74                  continue
75              if network_schedule.schedule[resource][schedule_pointers[
76                  resource]].start_time < next_start_time + candidate_task.
                        execution_time:
77                  return False
78          return True
79
80      def _future_minsep_violation(_tts: PacketGenerationTask) -> bool:
81          _reference_resource = _tts.path[1]
82          _schedule_except = network_schedule.schedule[_reference_resource]
83          offset = 0
84          while (
85                  schedule_pointers[_reference_resource] + offset < len(
86              _schedule_except)
87                  and
88                  _schedule_except[
89                      schedule_pointers[_reference_resource] + offset
90                  ].start_time < next_start_time + _tts.execution_time + _tts.minsep
91          ):
92              if _schedule_except[
93                  schedule_pointers[_reference_resource] + offset
94              ].task_id == _tts.task_id:
95                  return True
96              offset += 1
97          return False
98
99      def _past_minsep_violation(_tts: PacketGenerationTask) -> Tuple[
100         bool, float | None]:
101         _reference_resource = _tts.path[1]
102         _schedule_excerpt = network_schedule.schedule[_reference_resource]
```

```
        offset = 1 if schedule_pointers[_reference_resource] < len(
            _schedule_excerpt) else 2

        while (
                schedule_pointers[_reference_resource] - offset >= 0
                and (
                        _schedule_excerpt[
                            schedule_pointers[
                                _reference_resource
                            ] - offset
                        ].end_time > next_start_time - _tts.minsep
                        and not
                        np.isclose(
                            _schedule_excerpt[
                                schedule_pointers[
                                    _reference_resource
                                ] - offset
                            ].end_time,
                            next_start_time - _tts.minsep
                        )
                )
        ):

            if _schedule_excerpt[
                schedule_pointers[
                    _reference_resource
                ] - offset
            ].task_id == _tts.task_id:
                return True, _schedule_excerpt[
                                schedule_pointers[
                                    _reference_resource
                                ] - offset
                                ].end_time + _tts.minsep
            offset += 1
            if schedule_pointers[_reference_resource] - offset < 0:
                break

        return False, None

    if all(network_schedule.schedule[r][0].start_time > 0 for r in
           schedule_pointers.keys() if network_schedule.schedule[r]):
        next_start_time = min(network_schedule.schedule[r][0].end_time for r in
                              schedule_pointers.keys())

    while next_start_time < network_schedule.length:

        cycle_start_time = time.process_time()

        for index in _task_indices_to_inspect(round_robin_index):
            candidate_task = filling_class[index]
            # logger.debug(f'candidate_task: {candidate_task}')

            if (
                    not _resources_available(candidate_task)
                    or _future_minsep_violation(candidate_task)
                    or next_start_time > candidate_task.expiry_time
                    or next_start_time + candidate_task.execution_time >=
```

```
                        network_schedule.length
            ):
                continue

            past_minsep_violation, release_time = _past_minsep_violation(
                candidate_task)
            if past_minsep_violation:
                if release_time > next_start_time:
                    release_times_to_consider.add(release_time)
                continue

            network_schedule.schedule_packet_generation_attempt(
                candidate_task,
                next_start_time,
                False,
                schedule_pointers=schedule_pointers
            )

            round_robin_index = index + 1

        old_start_time = 0 + next_start_time

        next_start_time = _get_next_start_time()

        if next_start_time is None:
            return

        assert old_start_time < next_start_time

        # logger.debug(f"Completed cycle in time {time.process_time() -
            cycle_start_time:.5g}")

        if next_start_time is None:
            break

        # logger.debug(f"next start time is {next_start_time}")

        pass
```

Listing 5.2: Implementation of Algorithm 12 (BONUSROUNDROBIN) in Python. Note that rather than returning an updated network schedule, this subroutine simply updates the provided `NetworkSchedule` object.

Complete Scheduler

```
def complete_schedule_scheduler(
        filling_classes: SetOfFillingClasses,
        network_schedule: NetworkSchedule,
) -> Tuple[float, float]:
    logger = LogManager.get_resource_scheduler_logger()

    _start_minimal = time.process_time()
    logger.info("Computing␣Minimal␣Schedule")
    for filling_class in filling_classes.all_filling_classes:
        logger.debug(
            f"Starting␣to␣compute␣minimal␣schedule␣for␣filling␣class␣{filling_class}"
        )
        start_time = sum(fc.minimum_service_time for fc in
                         filling_classes.preceding_filling_classes[
```

```
                                    filling_class])
        minimal_allocation_scheduler(filling_class, network_schedule,
                                     start_time)

    minimal_time = time.process_time() - _start_minimal

    logger.debug(
        f"Completed minimal phase in time {time.process_time() - _start_minimal:.5g}"
    )

    _start_bonus = time.process_time()

    logger.info("Computing Bonus Schedule")
    for fc in filling_classes.all_filling_classes:
        logger.debug(
            f"Starting to compute bonus schedule for filling class {fc}")
        bonus_phase_round_robin_scheduler(fc, network_schedule)

    bonus_time = time.process_time() - _start_bonus

    logger.debug(
        f"Completed bonus phase in time {time.process_time() - _start_bonus:.5g}"
    )

    return minimal_time, bonus_time
```

Listing 5.3: Implementation of Algorithm **??** in Python.

ADMIT TASKS

```
def admit_new_tasks(filling_classes: SetOfFillingClasses, task_intake_object: Dict[
    int, List[PacketGenerationTask]], resources: List[int]) -> Dict[int, int]:
    """
    :param filling_classes: set of filling classes
    :param task_intake_object: task intake object in format {demand_id:[PGT]}
    :param resources: list of resources
    :return: list of accepted demands
    """

    logger = LogManager.get_admission_control_logger()

    available_time = {r : get_scheduling_interval_duration() for r in resources}
    for filling_class in filling_classes.all_filling_classes:
        for resource in filling_class.associated_resources:
            available_time[resource] -= filling_class.minimum_service_time

    accepted_demands = {}

    for demand in task_intake_object:
        logger.info(f"Considering demand {demand}")
        for pgt_to_admit in task_intake_object[demand]:
            logger.info(f"Considering PGT {pgt_to_admit}")
            try:
                filling_class = filling_classes.get_filling_class(pgt_to_admit)
            except FillingClassError:
                logger.info(f"Rejected PGT {pgt_to_admit} because no filling class
                    was found")
```

```
26                  continue
27
28              for resource in filling_class.associated_resources:
29                  available_time[resource] += filling_class.minimum_service_time
30
31              filling_class.add_packet_generation_task(pgt_to_admit)
32              if all(available_time[resource] - filling_class.minimum_service_time > 0
                    for resource in filling_class.associated_resources):
33
34
35                  accepted_demands[demand] = pgt_to_admit.task_id
36                  for resource in filling_class.associated_resources:
37                      available_time[resource] -= filling_class.minimum_service_time
38
39                  logger.debug(f"Remaining available time: {available_time}")
40                  logger.info(f"accepted demand: {demand} with PGT {pgt_to_admit}")
41
42                  break
43              else:
44                  logger.info(f"rejected PGT for insufficient time. Demand ID: {demand
                        }, PGT: {pgt_to_admit}")
45                  filling_class.remove(pgt_to_admit)
46                  for resource in filling_class.associated_resources:
47                      available_time[resource] -= filling_class.minimum_service_time
48      return accepted_demands
```

Listing 5.4: Implementation of Algorithm **??** in Python.

UPDATE FILLING CLASSES

```python
class CompletedDemands:
    expired_demands: List[int] = dataclasses.field(default_factory=list)
    removed_demands: List[int] = dataclasses.field(default_factory=list)
    terminated_demands: List[int] = dataclasses.field(default_factory=list)

    @property
    def all_removed_demands(self):
        return self.removed_demands + self.terminated_demands + self.expired_demands

def update_filling_classes(
        old_filling_classes: SetOfFillingClasses | None,
        path_partition: List[Set[Tuple[int, ...]]] | None,
        termination_buffer: Set[int],
        expiration_cutoff_time: float
) -> Tuple[
    SetOfFillingClasses,
    CompletedDemands,
]:

    completed_demands = CompletedDemands()

    if path_partition is not None:
        new_filling_classes = SetOfFillingClasses(path_partition)

        failed_tasks = new_filling_classes.add_packet_generation_tasks(
            old_filling_classes.all_tasks if old_filling_classes else [])
        completed_demands.removed_demands = [t.parent_demand_id for t in failed_tasks
            ]

    else:
        new_filling_classes = old_filling_classes

    expired, terminated = new_filling_classes.update_live_tasks(termination_buffer,
        expiration_cutoff_time)

    completed_demands.expired_demands = [task.parent_demand_id for task in expired]
    completed_demands.terminated_demands = [task.parent_demand_id for task in
        terminated]

    return new_filling_classes, completed_demands
```

Listing 5.5: Implementation of Algorithm 9 (UPDATEFILLINGCLASSES)

FILLING CLASSES

```
class FillingClass(List[PacketGenerationTask]):

    def __init__(self, path_partition: Set[Tuple[int, ...]]) -> None:
        super().__init__()
        self._path_partition: Set[Tuple[int, ...]] = path_partition
        self._associated_resources: Set[int] | None = None
        self._numbers_of_cycles: List[int] | None = None
        self._cycle_times: List[float] | None = None
        self._minimum_service_time: float | None = None

    def _calculate_associated_resources(self) -> None:
        self._associated_resources = set(
            [r for path in self._path_partition for r in path[1:-1]])

    @property
    def path_partition(self):
        return self._path_partition

    @property
    def associated_resources(self):
        if self._associated_resources is None:
            self._calculate_associated_resources()
        return self._associated_resources

    @property
    def associated_nodes(self) -> Set[int]:
        return {r for path in self._path_partition for r in path}

    def calculate_cycle_times(self):

        # self.sort(key=lambda task: task.minimum_allocation)

        self._cycle_times = [
            max([t.execution_time + t.minsep for t in self[i:]] + [
                sum(t.execution_time for t in self[i:])]) for i in
            range(len(self))]

        pass

    def calculate_numbers_of_cycles(self):

        # self.sort(key=lambda task: task.minimum_allocation)

        _list = [t.minimum_allocation for t in self]
        for i in range(len(self) - 1, 0, -1):
            _list[i] = _list[i] - _list[i - 1]

        _list[0] -= 1

        self._numbers_of_cycles = _list

    def calculate_minimum_service_time(self) -> None:
        self._minimum_service_time = sum(x * y for x, y in
                                         zip(self.numbers_of_cycles,
                                             self.cycle_times)) + sum(
```

```
56              t.execution_time for t in self)
57
58      def remove_expired_tasks(self, cutoff_time: float) -> List[
59          PacketGenerationTask]:
60          expired_tasks = [task for task in self if
61                           task.expiry_time <= cutoff_time]
62          self[:] = [task for task in self if task not in expired_tasks]
63          return expired_tasks
64
65      def remove_terminated_demands(self, terminated_demands: Iterable[int]) -> \
66      List[PacketGenerationTask]:
67          terminated_tasks = [task for task in self if
68                              task.parent_demand_id in terminated_demands]
69          self[:] = [task for task in self if task not in terminated_tasks]
70          return terminated_tasks
71
72      def update_live_tasks(
73              self,
74              terminated_demands: Set[int],
75              current_time: float
76      ) -> Tuple[
77          List[PacketGenerationTask],
78          List[PacketGenerationTask],
79      ]:
80
81          expired_tasks = self.remove_expired_tasks(current_time)
82          terminated_tasks = self.remove_terminated_demands(terminated_demands)
83
84          if self:
85              self.calculate_numbers_of_cycles()
86              self.calculate_cycle_times()
87              self.calculate_minimum_service_time()
88          else:
89              self._numbers_of_cycles = None
90              self._cycle_times = None
91              self._minimum_service_time = None
92
93          return expired_tasks, terminated_tasks
94
95      @property
96      def numbers_of_cycles(self):
97          if self._numbers_of_cycles is None or len(
98                  self._numbers_of_cycles) != len(self):
99              self.calculate_numbers_of_cycles()
100
101         return self._numbers_of_cycles
102
103     @property
104     def cycle_times(self):
105         if self._cycle_times is None or len(self._cycle_times) != len(self):
106             self.calculate_cycle_times()
107         return self._cycle_times
108
109     @property
110     def minimum_service_time(self):
111
112         if not self:
```

```
113             return 0
114
115         if self._minimum_service_time is None:
116             self.calculate_minimum_service_time()
117         return self._minimum_service_time
118
119     def add_packet_generation_task(self, task: PacketGenerationTask):
120         if not isinstance(task, PacketGenerationTask):
121             raise TypeError(
122                 f'Task must be of type PacketGenerationTask (provided type {type(task
                        )})')
123         _i = 0
124         while _i < len(self) and self[
125             _i].minimum_allocation < task.minimum_allocation:
126             _i += 1
127         self.insert(_i, task)
128         self._numbers_of_cycles = None
129         self._cycle_times = None
130         self._minimum_service_time = None
131
132     def clear(self):
133         super().clear()
134         self._numbers_of_cycles = None
135         self._cycle_times = None
136         self._minimum_service_time = None
137
138     def append(self, task: PacketGenerationTask):
139         if not isinstance(task, PacketGenerationTask):
140             raise TypeError(
141                 f"Task must be of type PacketGenerationTask (provided type {type(task
                        )})")
142         else:
143             super().append(task)
144             self._numbers_of_cycles = None
145             self._cycle_times = None
146             self._minimum_service_time = None
147
148     def remove(self, __value):
149         super().remove(__value)
150         self._numbers_of_cycles = None
151         self._cycle_times = None
152         self._minimum_service_time = None
```

Listing 5.6: Implementation of a single filling classes ϕ. Any properties of a filling class (e.g. `minimum_service_time`) which are `None` when called are calculated and stored. This means that subsequent calls to get the property only require $O(1)$ time rather than recomputing the value each time.

```
1  class SetOfFillingClasses:
2
3      def __init__(self, path_partition: List[Set[Tuple[int, ...]]]) -> None:
4
5          self._all_filling_classes: List[FillingClass] = [FillingClass(partition)
6                                                           for partition in
7                                                           path_partition]
8          self._all_filling_classes.sort(reverse=True)
9          self._preceding_filling_classes: Dict[
10             FillingClass, Set[FillingClass]] = {
```

```
            fc: {_fc for _fc in self._all_filling_classes if _fc > fc} if fc.
                path_partition else set() for fc in
            self._all_filling_classes}

        self._interface_filling_classes: List[FillingClass] | None = None
        self._local_area_filling_classes: List[FillingClass] | None = None
        self._backbone_filling_classes: List[FillingClass] | None = None

    def add_packet_generation_tasks(self, pgts_to_add: Iterable[
        PacketGenerationTask]) -> List[PacketGenerationTask]:
        failed_tasks: List[PacketGenerationTask] = []
        for pgt in pgts_to_add:
            try:
                self.get_filling_class(pgt).add_packet_generation_task(pgt)
            except FillingClassError:
                failed_tasks.append(pgt)

        return failed_tasks

    def update_live_tasks(
            self,
            terminated_demands: Set[int],
            current_time: float
    ) -> Tuple[
        List[PacketGenerationTask],
        List[PacketGenerationTask],
    ]:
        expired_tasks = []
        terminated_tasks = []

        for filling_class in self.all_filling_classes:
            _ex, _term = filling_class.update_live_tasks(terminated_demands,
                                                         current_time)
            expired_tasks.extend(_ex)
            terminated_tasks.extend(_term)

        return expired_tasks, terminated_tasks

    def get_filling_class(self, pgt: PacketGenerationTask) -> FillingClass:
        for filling_class in self._all_filling_classes:
            if pgt.path in filling_class.path_partition:
                return filling_class

        raise FillingClassError("No known filling class for pgt")
```

Listing 5.7: Implementation of the set of filling classes Φ. Any properties of a filling class (e.g. `minimum_service_time`) which are `None` when called are calculated and stored. This means that subsequent calls to get the property only require $O(1)$ time rather than recomputing the value each time.

NETWORK SCHEDULES

```
class PacketGenerationAttempt(ArqonDataRecord):
    src: int
    dst: int
    start_time: float
    end_time: float
```

```
    task_id: int
    demand_id: Optional[int]
    pga_id: Optional[int]
    is_minimal: bool

class ResourceSchedule(List[PacketGenerationAttempt]):
    def __init__(self, resource_id: int, seq: tuple = ()):
        super().__init__(seq)
        self._resource_id = resource_id

    @property
    def resource_id(self):
        return self._resource_id



    def add_pga(self, __object: PacketGenerationAttempt, pointer: Optional[int] =
         None):
        """
        re-written append method to ensure time-ordering of PGAs.
        :param __object: PacketGenerationAttempt to add to resource schedule.
        :param pointer: index of PGA preceding requested time
        """
        if not isinstance(__object, PacketGenerationAttempt):
            raise TypeError(f"Only␣type␣PacketGenerationAttempt␣is␣allowed.␣(supplied
                ␣{type(__object)})")

        else:
            if not self:
                super().append(__object)
                return

            if pointer is None:
                pointer = _bs(self, __object.end_time)

                if pointer > 0:
                    self.insert(pointer + 1, __object)
                elif self[pointer].end_time < __object.end_time:
                    self.insert(1, __object)
                else:
                    self.insert(0, __object)

            else:
                if pointer == 0:
                    self.insert(pointer, __object)
                elif pointer > 0:
                    self.insert(pointer, __object)
                elif pointer < 0:
                    super().append(__object)
        return {
            "node": self.resource_id,
            "packet_generation_attempts": [pga.to_dict() for pga in self]
        }



class NetworkSchedule:
```

```
61
62      def __init__(self, start_time: float, length: float, resources: List[int],
            identifier: int | None = None):
63
64          self._schedule: Dict[int, ResourceSchedule] = {r: ResourceSchedule(r) for r
                in resources}
```

Listing 5.8: Implementation of the network schedule.

REFERENCES

[1] P. Arright and L. Salvail, *Blind quantum computation,* International Journal of Quantum Information **04**, 883 (2006).

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

[3] A. M. Childs, *Secure assisted quantum computation,* Quantum Information and Computation **5** (2005), 10.26421/QIC5.6.

[4] 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.

[5] A. K. Ekert, *Quantum cryptography based on bell's theorem,* Phys. Rev. Lett. **67**, 661 (1991).

[6] M. Hasanpour, S. Shariat, P. Barnaghi, S. A. Hoseinitabatabaei, S. Vahid, and R. Tafazolli, *Quantum load balancing in ad hoc networks,* Quantum Information Processing **16**, 148 (2017).

[7] D. Ding and L. Jiang, *Coordinating Decisions via Quantum Telepathy,* (2024).

[8] J. Tucker, P. Strange, P. Mironowicz, and J. Quintanilla, *Quantum-assisted rendezvous on graphs: explicit algorithms and quantum computer simulations,* New Journal of Physics **26**, 093038 (2024).

[9] C. Brukner, N. Paunkovic, T. Rudolph, and V. Vedral, *Entanglement-assisted orientation in space,* (2005), arXiv:quant-ph/0509123 [quant-ph] .

[10] G. Viola and P. Mironowicz, *Quantum strategies for rendezvous and domination tasks on graphs with mobile agents,* Physical Review A **109** (2024), 10.1103/physreva.109.042201.

[11] S. Tani, H. Kobayashi, and K. Matsumoto, *Exact Quantum Algorithms for the Leader Election Problem,* ACM Transactions on Computation Theory **4**, 1 (2012).

[12] P. Kómár, E. M. Kessler, M. Bishof, L. Jiang, A. S. Sørensen, J. Ye, and M. D. Lukin, *A quantum network of clocks,* Nature Physics **10**, 582–587 (2014).

[13] D. Gottesman, T. Jennewein, and S. Croke, *Longer-baseline telescopes using quantum repeaters,* Phys. Rev. Lett. **109**, 070503 (2012).

[14] T. R. Beauchamp, H. Jirovská, S. Gauthier, and S. Wehner, *A modular quantum network architecture for integrating network scheduling with local program execution,* IEEE Transactions on Quantum Engineering , 1 (2025).

[15] C. Delle Donne, M. Iuliano, B. van der Vecht, G. M. Ferreira, H. Jirovska, T. J. W. van der Steenhoven, A. Dahlberg, M. Skrzypczyk, D. Fioretto, M. Teller, P. Filippov, A. R.-P. Montblanch, J. Fischer, H. B. van Ommen, N. Demetriou, D. Leichtle, L. Music, H. Ollivier, I. te Raa, W. Kozlowski, T. H. Taminiau, P. Pawełczak, T. E. Northup, R. Hanson, and S. Wehner, *An operating system for executing applications on quantum network nodes,* Nature **639**, 321–328 (2025).

[16] B. van der Vecht, A. T. Yücel, H. Jirovska, and S. Wehner, *Qoala: an application execution environment for quantum internet nodes,* (2025), 2502.17296 [quant-ph] .

[17] L. Peterson and B. Davie, *Computer Networks: A Systems Approach* (Elsevier, 2012) Chap. 2.5.

[18] 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* (2019) pp. 159–173.

[19] H. Zimmermann, *Osi reference model - the iso model of architecture for open systems interconnection,* IEEE Transactions on Communications **28**, 425 (1980).

[20] 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 **8**, 1 (2022).

[21] W.-Z. Liu, Y.-B. Zhou, J.-P. Chen, B. Wang, A. Teng, X.-W. Han, G.-C. Liu, Z.-J. Zhang, Y. Yang, F.-G. Liu, C.-H. Xue, B.-W. Yang, J. Yang, C. Zeng, D.-R. Pan, M.-Y. Zheng, X. Zhang, S. Cao, Y.-Z. Zhen, Y. Xiao, H. Li, L. You, X. Ma, Q. Zhao, F. Xu, Y. Wang, Y. Wan, Q. Zhang, and J.-W. Pan, *Long-lived remote ion-ion entanglement for scalable quantum repeaters,* Nature (2026), 10.1038/s41586-026-10177-4.

[22] C. G. Davis, *An experimental pulse code modulation system for short-haul trunks,* The Bell System Technical Journal **41**, 1 (1962).

[23] A. S. Tanenbaum, *Computer Networks*, 4th ed. (Prentice Hall PTR, Upper Saddle River, New Jersey, 2003) p. 912.

[24] R. Bhardwaj and B. Jabbari, *Isdn frame relay and its congestion control,* International Journal of Satellite Communications **9**, 363 (1991), https://onlinelibrary.wiley.com/doi/pdf/10.1002/sat.4600090513 .

[25] A. Waters and K. Ab-Hamid, *Congestion control for frame relay networks,* in *IEE Colloquium on Frame Relay* (1992) pp. 3/1–3/4.

[26] R. Jain, *Congestion control and traffic management in atm networks: Recent advances and a survey,* Computer Networks and ISDN Systems **28**, 1723 (1996).

[27] K. Steenhaut, K. Degieter, W. Brissinck, and E. Dirkx, *Scheduling and admission control policies: A case study for ATM,* Signaling and Management of ATM Networks, **29**, 539.

[28] C. Cicconetti, M. Conti, and A. Passarella, *Request Scheduling in Quantum Networks,* IEEE Transactions on Quantum Engineering **2**, 2 (2021).

[29] A. Pirker and W. Dür, *A quantum network stack and protocols for reliable entanglement-based networks,* New Journal of Physics **21**, 033003 (2019).

[30] 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),* 2023 32nd International Conference on Computer Communications and Networks (ICCCN) , 1 (2023).

[31] R. Van Meter, T. Satoh, T. D. Ladd, W. J. Munro, and K. Nemoto, *Path Selection for Quantum Repeater Networks,* Networking Science **3**, 82 (2013).

[32] W. Dür, H.-J. Briegel, J. I. Cirac, and P. Zoller, *Quantum repeaters based on entanglement purification,* Physical Review A **59**, 169 (1999).

[33] 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)* (2022) pp. 341–352.

[34] L. Bacciottini, M. G. D. Andrade, S. Pouryousef, E. A. V. Milligen, A. Chandra, N. K. Panigrahy, N. S. V. Rao, G. Vardoyan, and D. Towsley, *Leveraging internet principles to build a quantum network,* IEEE Network , 1 (2025).

[35] S.-y. Yu, E. Perego, J. Phillips, Y.-W. Cheah, P. Umesh, G. Gao, J. Liu, E. Kissel, M. Bregar, K. Sun, Q. Wu, R. Valivarthi, E. Saglamyurek, W. Wu, M. Spiropulu, H. Häffner, and I. Monga, *A two-level control framework for quantum networks,* in *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*, Vol. 01 (2025) pp. 1302–1311.

[36] M. Skrzypczyk and S. Wehner, *An Architecture for Meeting Quality-of-Service Requirements in Multi-User Quantum Networks,* (2021).

[37] T. Sekimoto and J. Puente, *A satellite time-division multiple-access experiment,* IEEE Transactions on Communication Technology **16**, 581 (1968).

[38] M. Ruf, N. H. Wan, H. Choi, D. Englund, and R. Hanson, *Quantum networks based on color centers in diamond,* Journal of Applied Physics **130**, 070901 (2021).

[39] 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 **372**, 259 (2021).

[40] 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. **130**, 050803 (2023).

[41] 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 **3**, 538 (2007).

[42] J. P. Covey, H. Weinfurter, and H. Bernien, *Quantum networks with neutral atom processing nodes,* npj Quantum Information **9**, 1 (2023).

[43] 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 **122** (2016), 10.1007/s00340-015-6299-2.

[44] 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 **438**, 828 (2005).

[45] 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 **316**, 1316 (2007).

[46] 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 **132** (2024), 10.1103/physrevlett.132.150604.

[47] 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 **5**, 186 (2022).

[48] S. Gauthier, G. Vardoyan, and S. Wehner, *An architecture for control of entanglement generation switches in quantum networks,* IEEE Transactions on Quantum Engineering **4**, 1 (2023).

[49] S. Gauthier, T. Vasantam, and G. Vardoyan, *On-demand resource allocation for a quantum network hub,* IEEE Transactions on Quantum Engineering **7**, 1 (2026).

[50] H. K. Beukers, M. Pasini, H. Choi, D. Englund, R. Hanson, and J. Borregaard, *Remote-entanglement protocols for stationary qubits with photonic interfaces,* PRX Quantum **5** (2024), 10.1103/prxquantum.5.010202.

[51] C. Jones, D. Kim, M. T. Rakher, P. G. Kwiat, and T. D. Ladd, *Design and analysis of communication protocols for quantum repeater networks,* New Journal of Physics **18**, 083015 (2016).

[52] C. Cabrillo, J. I. Cirac, P. García-Fernández, and P. Zoller, *Creation of entangled states of distant atoms by interference,* Phys. Rev. A **59**, 1025 (1999).

[53] L.-M. Duan, M. D. Lukin, J. I. Cirac, and P. Zoller, *Long-distance quantum communication with atomic ensembles and linear optics,* Nature **414**, 413 (2001).

[54] S. L. Braunstein and A. Mann, *Measurement of the Bell operator and quantum teleportation,* Physical Review A **51**, R1727 (1995).

[55] M. Michler, K. Mattle, H. Weinfurter, and A. Zeilinger, *Interferometric Bell-state analysis,* Physical Review A **53**, R1209 (1996).

[56] P. Walther and A. Zeilinger, *Experimental realization of a photonic Bell-state analyzer,* Physical Review A **72**, 010302 (2005).

[57] S. Daiss, S. Langenfeld, S. Welte, E. Distante, P. Thomas, L. Hartung, O. Morin, and G. Rempe, *A quantum-logic gate between distant quantum-network modules,* Science **371**, 614 (2021), https://www.science.org/doi/pdf/10.1126/science.abe3150 .

[58] C. T. Nguyen, D. D. Sukachev, M. K. Bhaskar, B. Machielse, D. S. Levonian, E. N. Knall, P. Stroganov, R. Riedinger, H. Park, M. Lončar, and M. D. Lukin, *Quantum network nodes based on diamond qubits with an efficient nanophotonic interface,* Phys. Rev. Lett. **123**, 183602 (2019).

[59] 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. **98**, 190503 (2007).

[60] N. Sangouard, C. Simon, H. de Riedmatten, and N. Gisin, *Quantum repeaters based on atomic ensembles and linear optics,* Rev. Mod. Phys. **83**, 33 (2011).

[61] N. Sangouard, R. Dubessy, and C. Simon, *Quantum repeaters based on single trapped ions,* Phys. Rev. A **79**, 042340 (2009).

[62] S. Maiti, G. Avis, S. Kar, and S. Wehner, *Requirements for teleportation in an intercity quantum network,* (2026), arXiv:2602.04869 [quant-ph] .

[63] H.-J. Briegel, W. Dür, J. I. Cirac, and P. Zoller, *Quantum repeaters: The role of imperfect local operations in quantum communication,* Physical Review Letters **81**, 5932–5935 (1998).

[64] A. G. Inesta, G. Vardoyan, L. Scavuzzo, and S. Wehner, *Optimal entanglement distribution policies in homogeneous repeater chains with cutoffs,* npj Quantum Information **9**, 1 (2023).

[65] K. Goodenough, T. Coopmans, and D. Towsley, *On noise in swap asap repeater chains: exact analytics, distributions and tight approximations,* Quantum **9**, 1744 (2025).

[66] W. H. Zurek, *Environment-induced superselection rules,* Phys. Rev. D **26**, 1862 (1982).

[67] W. G. Unruh, *Maintaining coherence in quantum computers,* Phys. Rev. A **51**, 992 (1995).

[68] C. Liu, Z. Dutton, C. H. Behroozi, and L. V. Hau, *Observation of coherent optical information storage in an atomic medium using halted light pulses,* Nature **409**, 490 (2001).

[69] D. F. Phillips, A. Fleischhauer, A. Mair, R. L. Walsworth, and M. D. Lukin, *Storage of light in atomic vapor,* Physical Review Letters **86**, 783 (2001).

[70] R. Zhao, Y. O. Dudin, S. D. Jenkins, C. J. Campbell, D. N. Matsukevich, T. A. B. Kennedy, and A. Kuzmich, *Long-lived quantum memory,* Nature Physics **5**, 100 (2009).

[71] 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 **497**, 86 (2013).

[72] 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 **558**, 268 (2018).

[73] 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 **607**, 69 (2022).

[74] 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*, OSDI '06 (USENIX Association, USA, 2006) p. 89–102.

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

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

[77] 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 **103**, 14 (2015).

[78] 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*, Request for Comments RFC 5810 (Internet Engineering Task Force, 2010).

[79] 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 **38**, 69 (2008).

[80] 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.

[81] T. Beauchamp, S. Gauthier, and S. Wehner, *Code and data for for arqon: A suite of control applications enabling reliable quantum networks,* (2026).

[82] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters, *Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels,* Phys. Rev. Lett. **70**, 1895 (1993).

[83] M. Horodecki, P. Horodecki, and R. Horodecki, *General teleportation channel, singlet fraction, and quasidistillation,* Phys. Rev. A **60**, 1888 (1999).

[84] D. Deutsch, A. Ekert, R. Jozsa, C. Macchiavello, S. Popescu, and A. Sanpera, *Quantum privacy amplification and the security of quantum cryptography over noisy channels,* Physical Review Letters **77**, 2818–2821 (1996).

[85] C. H. Bennett, G. Brassard, S. Popescu, B. Schumacher, J. A. Smolin, and W. K. Wootters, *Purification of noisy entanglement and faithful teleportation via noisy channels,* Physical Review Letters **76**, 722–725 (1996).

[86] S. Abruzzo, S. Bratzik, N. K. Bernardes, H. Kampermann, P. van Loock, and D. Bruß, *Quantum repeaters and quantum key distribution: Analysis of secret-key rates,* Physical Review A **87** (2013), 10.1103/physreva.87.052315.

[87] 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 **5**, 1 (2024).

[88] A. Gruber, A. Dräbenstedt, C. Tietz, L. Fleury, J. Wrachtrup, and C. von Borczyskowski, *Scanning confocal optical microscopy and magnetic resonance on single defect centers,* Science **276**, 2012 (1997).

[89] L. Childress and R. Hanson, *Diamond NV centers for quantum computing and quantum networks,* MRS Bulletin **38**, 134 (2013).

[90] M. V. G. Dutt, L. Childress, L. Jiang, E. Togan, J. Maze, F. Jelezko, A. S. Zibrov, P. R. Hemmer, and M. D. Lukin, *Quantum register based on individual electronic and nuclear spin qubits in diamond,* Science **316**, 1312 (2007).

[91] J. I. Cirac and P. Zoller, *Quantum computations with cold trapped ions,* Phys. Rev. Lett. **74**, 4091 (1995).

[92] B. B. Blinov, D. L. Moehring, L.-M. Duan, and C. Monroe, *Observation of entanglement between a single trapped atom and a single photon,* Nature **428**, 153 (2004).

[93] V. Krutyanskiy, M. Canteri, M. Meraner, J. Bate, V. Krcmarsky, J. Schupp, N. Sangouard, and B. P. Lanyon, *Telecom-wavelength quantum repeater node based on a trapped-ion processor,* Phys. Rev. Lett. **130**, 213601 (2023).

[94] J. I. Naus, *Approximations for Distributions of Scan Statistics,* Journal of the American Statistical Association **77**, 177 (1982).

[95] W. Hoeffding, *Probability inequalities for sums of bounded random variables,* Journal of the American Statistical Association **58**, 13 (1963).

[96] C. Jones, D. Kim, M. T. Rakher, P. G. Kwiat, and T. D. Ladd, *Design and analysis of communication protocols for quantum repeater networks,* New Journal of Physics **18**, 083015 (2016).

[97] G. Vardoyan, S. Guha, P. Nain, and D. Towsley, *On the exact analysis of an idealized quantum switch,* SIGMETRICS Perform. Eval. Rev. **48**, 79–80 (2021).

[98] T. Coopmans, S. Brand, and D. Elkouss, *Improved analytical bounds on delivery times of long-distance entanglement,* Physical Review A **105** (2022), 10.1103/physreva.105.012608.

[99] P. Baran, *On distributed communications networks,* IEEE Transactions on Communications Systems **12**, 1 (1964).

[100] S. E. Alm, *On the distributions of the scan statistics of a poisson process,* , 1.

[101] J. Glaz, J. I. Naus, and S. Wallenstein, *Scan statistics*, Springer series in statistics (Springer, New York, 2001).

[102] J. Glaz and N. Balakrishnan, eds., *Scan Statistics and Applications* (Birkhäuser Boston).

6

CONCLUSION

6.1. SUMMARY OF RESULTS

- **Metropolitan hub control architecture:** We have presented a versatile control architecture for a metropolitan hub, also referred to as an Entanglement Generation Switch (EGS). This control architecture can be adapted by specification of different request handling mechanisms, service termination mechanisms, and load balancing protocols. Critically, load balancing mechanisms support scaling a metropolitan hub to accommodate 100 or more end nodes simultaneously by mitigating the competition for resources that results when many pairs of end nodes demand service concurrently. In Chapters 2 and 3 we proposed and analyzed two separate implementations of this control architecture. In both implementations demands for service are repeatedly submitted by pairs of end nodes to the metropolitan hub according to some locally determined rate. In Chapter 2 this rate has physical meaning as a demanded rate of successful entanglement generation. In Chapter 3 the rate simply determines the inter-arrival times between repeated demands for service.

  - The request handling mechanism introduced in Chapter 2 is based on a queuing system with priority assigned to requests depending on queue length. In contrast, the implementation in Chapter 3 introduced a queue-less system where requests are immediately accepted or rejected based on resource availability.
  - To handle service termination, we defined mechanisms in both Chapter 2 and in Chapter 3 where service terminates according to a relative expiry time based on the service start time, or else is possibly terminated early due to successful entanglement generation.
  - To mitigate competition for resources and prevent the time between service requests and service delivery from growing endlessly, in Chapter 2 we proposed a rate control protocol in which pairs of end nodes monitor and modify

their requested rates of entanglement generation. These modifications are based on locally defined conditions and a feedback signal indicative of the current contention for resource use, communicated by the metropolitan hub to end nodes. We proposed this protocol in the context of a metropolitan hub employing queue based request handling. In [1] we patented this protocol in extended generality, covering also implementations in which the metropolitan hub employs a queue-less system of request handling.

In the context of a long-distance quantum network, a metropolitan hub is intended to function as one of several sub-systems. The control architecture for a metropolitan hub introduced here can be adapted to comply with a wide variety of full quantum network control architectures by supplanting the protocols defined in Chapters 2 and 3 with alternative protocols implementing each of these mechanisms.

- **Long-distance network control architecture:** We formulated a centrally controlled architecture for long-distance entanglement based quantum networks with the goal of facilitating successful execution of applications on independently programmable end nodes (Chapter 4). We developed a demand based service model which defines a clear separation between end nodes and the internal components of the network. End nodes encode their application derived requirements as demands for packets of entanglement. In response to end node demands, a central controller periodically produces and distributes network schedules allocating the the use of all internal nodes along end-to-end routes for definite periods of time called Packet Generation Attempts (PGAs). The duration of PGAs and the number scheduled to service each demand are calculated based on the expected rate of end-to-end entanglement generation and an application derived deadline before which all packets of entanglement should be successfully generated. This network scheduling mechanism combats the technological challenge arising from limited maximum rates of entanglement generation, as compared to maximum entanglement storage times.

- **Deployment ready implementation:** In Chapter 4 we defined an initial implementation of our long-distance network control architecture, including policies for admission control and network schedule computation. In the context of a limited metropolitan network, this implementation failed to robustly resolve the contention for network resources that arises when multiple demands for entanglement compete for the same network resources.

  In Chapter 5 we defined a set of requirements for a reliable quantum network. These requirements encode necessary features of an implementation suitable for deployment on the QIA prototype network. We then presented the Arqon suite of control applications and defined a concrete service agreement that Arqon issues to end nodes whenever a new demand is accepted for service. We demonstrated both analytically and numerically that an implementation of our network architecture equipped with the Arqon control applications upholds issued service agreements and satisfies all reliability requirements. Critically, an admission control applica-

tion prioritizes serving existing demands over serving newly arrived demands and systematically prevents resource contention in the set of arriving demands from affecting the network schedule.

The results of this thesis pave the way for demonstrating multi-user access to an entanglement-based network. These control architectures will directly enable the successful deployment of the QIA prototype network, illustrated in Figure 1.5 of the Introduction.

Two complementary branches of activities, which are directly connected to the research in this thesis, are needed to prepare the deployment of this prototype network:

1. the final design for every network sub-system and the interfaces between sub-systems must be confirmed, and each sub-system must be built and/or implemented individually;

2. the individual sub-systems must be integrated into a complete network by connecting them across their designed interfaces, and the functionality of this integrated system must be validated through integration testing.

In complement to defining a control architecture for a metropolitan hub, the research contained in this thesis led to the identification of hardware requirements for a metropolitan hub (Section 6.2), thereby contributing to branch 1.

One strategy to complete the integration activities mandated by branch 2 is to separate the integration of all sub-systems into many smaller stages, in each of which a small number of sub-systems are integrated together. This strategy allows isolating the validation of a few interfaces at a time, simplifying fault diagnosis or the identification of required optimizations or upgrades. In Section 6.3 we describe a planned experimental demonstration that will validate the integration of the Arqon control architecture (an implementation of the control architecture of Chapter 4 equipped with the Arqon control applications of Chapter 5) with two programmable quantum end nodes. The preparation for this integration experiment is the final contribution of this thesis, and leaves us one step closer to demonstrating multi-user access to entanglement based quantum networks.

6.2. REQUIRED HARDWARE FOR A METROPOLITAN HUB

Metropolitan hubs provide multi-user access to central infrastructure for quantum networking, enabling multiple quantum nodes to share the infrastructure required for entanglement generation and related quantum information protocols, such as remote state preparation. The specific equipment requirements depend on the physical layer protocols that the hub needs to support.

For the QIA systems tests, the following physical layer protocol considerations apply:

- Two trapped ion based processing nodes [2, 3] connected via the metropolitan hub in one metropolitan network must be able to successfully execute a midpoint detection based heralded entanglement generation protocol (see [4] for protocol classifications).

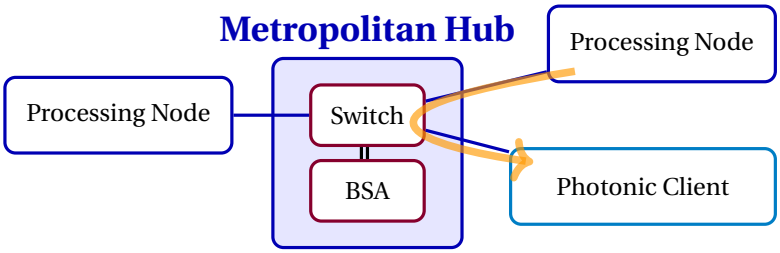


Figure 6.1: High level overview of the access sharing architecture for a metropolitan hub in the QIA prototype network. The switch either routes the inputs from two processing nodes to the two input ports of a BSA, or else the switch routes the input of a one processing node to an output port connected to the measurement based photonic client.

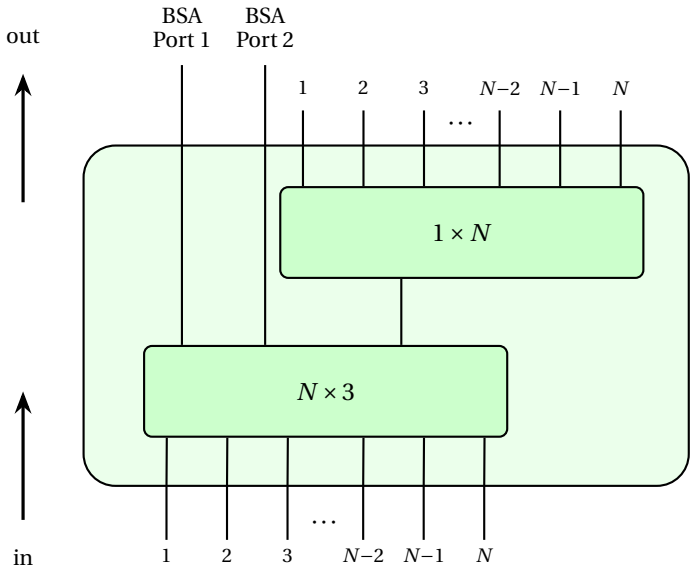


Figure 6.2: Sketch of the internal design of a switch implementing the connectivity required for the QIA metropolitan hub. N input ports connect to an $N \times 3$ switch which connects any 3 input ports to: the first and second input ports of the BSA, and the single input port of a $1 \times N$ switch. The second layer $1 \times N$ switch connects it's single input port to any of the N output ports. Many variations of this design are possible.

- One processing node and one measurement based photonic client [5, 6] connected via the metropolitan hub must be able to successfully execute a remote state preparation protocol. In this protocol the processing node must emit photons, the metropolitan hub must route them to the measurement client, and the measurement client must measure received photons.

- To mitigate photon transmission loss, frequency conversion must be employed to convert the photons emitted by the processing nodes from their natural emission frequency to a pre-specified telecom frequency. This requirement is significant because the processing nodes will be connected to the metropolitan hub via links of approximately 25 km, a distance scale over which signal attenuation due to photon transmission loss in optical fiber is substantial.

Based on these physical layer protocol requirements, here we identify required hardware and constraints on the interfaces between devices for the metropolitan hub in the QIA prototype network. The following inventory of requirements is partially informed by the systems architecture of the midpoint node in [7], which successfully supported midpoint detection based heralded entanglement generation between two fully independent end nodes located 10 km apart, by providing shared access to infrastructure. The core equipment requirements are:

Bell-State Analyzer(s) (BSA): These are a type of entanglement generation interface enabling midpoint-detection-based entanglement generation protocols [4, 8–11]. The BSA performs joint measurements on photons from separate nodes, projecting them into an entangled state. This Bell-state measurement procedure is also known as an entanglement swap.

Clock Signal Distribution: Independent nodes use the clock signal to enable precise timing synchronization with partner nodes. This is essential for enabling coordination of quantum operations between distributed nodes with nanosecond precision, a prerequisite of midpoint detection based entanglement generation protocols [4, 8, 12].

Environmental Controls for Nodes: Temperature, humidity, and vibration control systems are necessary to maintain stable conditions. Environmental fluctuations can significantly impact protocol success rates. Real time monitoring and feedback control can mitigate these effects and support continuous device operation [7, 13].

Quantum Frequency Conversion (QFC) Support: The metropolitan hub must provide equipment supporting nodes in converting their native photon emission frequencies to a precisely defined telecom frequency. This support entails the following equipment requirements [7, 14]:

- *Reference Frequency Source:* Provides a stable frequency reference for the conversion process. Critical for aligning the frequencies of converted photons from independent processing nodes.

- *Interferometer:* Enables measurement of frequency differences between the reference light and the converted photons from nodes.
- *Feedback Mechanism:* Provides real-time adjustments to individual nodes. Ensures stable alignment across the quantum network despite environmental fluctuations.

Another core requirement of a metropolitan hub is an access sharing solution, providing a method for sharing access to the BSA among multiple processing nodes, as well as linking measurement based photonic clients to processing nodes. For the purposes of the QIA prototype network, there are two suitable options: fast optical switching, or "logical" switching, where there is no physical reconfiguration and switching is implemented only in software or at a protocol level, see e.g. [15].

Fast Optical Switch:

- *Pros:* Scalable to many nodes, physically routes optical paths.
- *Cons:* Higher cost.
- *Best for:* Large-scale metropolitan networks with more than two nodes.

Logical Switching:

- *Pros:* Minimal hardware requirements (i.e. no active switching mechanism needed).
- *Cons:* Limited scalability, not suitable for many-node scenarios.
- *Best for:* Few-node implementations, initial system tests with possible network re-configuration between tests.

When there are measurement-based photonic clients, the access sharing switching architecture of the metropolitan hub must provide configurable paths that can either route processing nodes to the BSA or bypass it entirely, allowing photons to be transmitted from one processing node to the measurement-based client. An overview of the required optical routing is illustrated in Figure 6.1. There are many possibilities for the internal architecture of an optical switch implementing the required connectivity. In Figure 6.2 we sketch a possible design for an N input switch which can connect any two devices to the first and second port of a BSA, and simultaneously connect the optical paths of two of the remaining $N-2$ devices. The input devices may be processing nodes or photonic clients. In a minimal implementation of this design, there could be N input ports – with N the number of processing nodes – and only 3 output ports, one for each of the BSA input ports and one for the measurement based photonic client. In the minimal implementation the additional $1 \times N$ switching block could then be totally removed from the switching design.

The specific midpoint detection heralded entanglement generation protocol that will be used between processing nodes has implications on the additional equipment requirements of the metropolitan hub. If a protocol is used which makes use of a single-click detection method [12, 16], then the metropolitan hub requires equipment supporting a phase stabilization architecture, such as that employed in [7] and further detailed in [17].

If a protocol is used that makes use of a double-click detection method [2, 8, 18], then no phase stabilization is necessary, nor additional equipment. At this time, QIA is exploring the technical feasibility of both double and single click options.

6.3. ARQON INTEGRATION AND EXPERIMENTAL EVALUATION

The system architecture of a two node quantum network controlled by Arqon via a Software-Defined Networking (SDN) controller is illustrated in Figure 6.3. The two programmable quantum end nodes indicated in the system architecture belong to the Quantum Network Explorer [19]. Each of these end nodes includes a two qubit processor based on the NV^- color center in diamond [12, 18, 20, 21]. At each node one of the two qubits is the electron spin of the NV^- color center, known also as a communication qubit because it provides the spin-photon interface enabling a remote entanglement generation protocol. The second qubit at each node is a C^{13} atom in the diamond lattice, which provides a quantum memory with a storage time that is long compared to the duration of single qubit gates. Universal two qubit quantum logic is possible in each node, exploiting control over the electron and C^{13} qubits and their interactions.

6.3.1. COMPONENTS OF THE INTEGRATED SYSTEM

- **Arqon:** A real-time software implementation of Arqon based on the Implementation Section of Chapter 5.

- **SDN Controller:** A custom SDN controller mediates all communication between end nodes and Arqon. The SDN controller is developed by the Stack Team at TU Delft. The required interfaces between the SDN controller and Arqon were identified in a collaboration between the Stack Team and the Arqon team, during the design phase of Arqon and its real-time software implementation.

- **End Node Runtime:** Logically, the node runtime is partitioned into a Classical Processing System (CPS), a Quantum Processing System (QPS), and a Node Scheduler. The node runtime system of each of end nodes 1 and 2 is an upgraded version of the quantum network node operating system QNodeOS [22]. To support this integration effort, QNodeOS has been upgraded to support submission of a pre-programmed application demand to the SDN controller. It has also been upgraded to enhance its ability to comply with network schedules. Previously, it was possible to start entanglement generation according to a network schedule, but not to stop according to the end of the scheduled time. Now, it is possible to start and stop entanglement generation according to a network schedule.

  - **CPS**: The CPS manages classical computing resources (for example CPU, classical memory and networking). It contains an SDN-agent which manages all communication with the SDN controller, including submission of application demands and retrieval of network schedules.
  - **QPS**: The QPS is responsible for executing quantum operations including entanglement generation, gates, and measurements on quantum hardware. In each of the end nodes in this integrated system, the quantum hardware is a

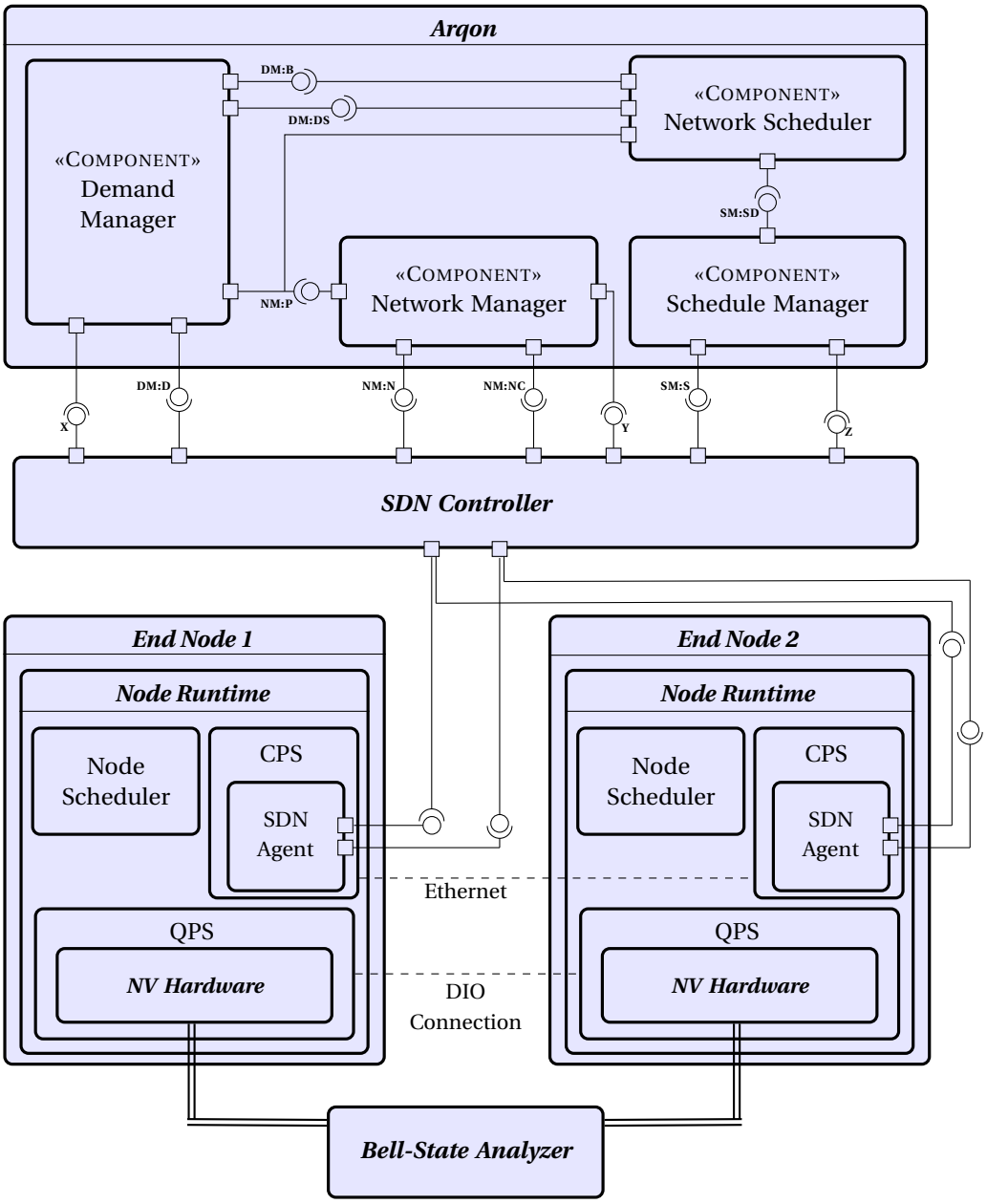


Figure 6.3: A two node quantum network controlled by Arqon via a Software-Defined Networking (SDN) controller.

two qubit processor based on the NV color center in diamond [12, 18, 20, 21]. The QPSs of the two end nodes share a common clock, enabling the precise timing synchronization required for entanglement generation.

  - **Node Scheduler**: The Node Scheduler is responsible for starting the execution of programs and timing the execution of application instructions on the QPS. In particular, entanglement generation instructions must be timed according to the packet generation attempts included in the network schedule received from Arqon.
  - *Ethernet connection*: The Ethernet connections between the CPSs of the end nodes enables any classical communication required by the execution of application programs. For example, a program for teleporting an arbitrary quantum state from end node 1 to 2 requires the CPS of node 1 to communicate two measurement outcomes to the CPS of end node 2; the CPS of end node 2 uses the received measurement outcomes as an input to an instruction which can then be executed on the QPS.
  - *Digital connection*: There is a digital input/output (DIO) connection between the QPSs of the two end nodes. This connection is used for low level communication, including agreeing on a common epoch (with respect to a common clock) for triggering batches of entanglement generation attempts by sending voltage pulses between the QPSs.

- **Bell-State Analyzer**: The two end nodes are connected to a Bell-state analyzer by optical fiber. This entanglement generation interface supports heralded entanglement generation between the two end nodes following the link layer “entanglement generation protocol" defined in [23, 24]. The connection between each end node and the Bell-state analyzer includes an optical fiber based quantum channel and a gigabit Ethernet based classical channel. The quantum channel transmits single photons emitted by the NV hardware to the Bell-State Analyzer and the classical channel communicates entanglement heralding success/failure outcomes from the Bell-state analyzer to the end nodes.

6.3.2. TEST APPLICATIONS

Two test applications are defined for the experimental evaluation of the integrated system in Figure 6.3:

1. *Teleportation:* In this application a single entangled pair between two nodes is consumed as a resource to facilitate the transfer (teleportation) of quantum state of an additional qubit from one node to the other. If the entangled pair consumed is a Bell-state and there are no additional sources of error, then the transfer is perfect. In the case that the entangled pair has fidelity $F < 1$ to a Bell-state, then the teleported state will have fidelity [25]

$$F_{\text{teleported}} = \frac{2F+1}{3} \tag{6.1}$$

to the original state. This application is a quantum networking primitive which enables many applications of increased complexity, e.g. [26–32]. An instance of this

application is said to succeed if the final teleported state has fidelity $F_{\text{teleportation}} > 0.66$ to the original state, where 0.66 is known as the classical bound (obtained if $F = 0.5$ in (6.1)).

This application requires only a single entangled pair and has a relatively low minimum required fidelity. For these reasons, it is possible to execute a large number of instances in a short period of time, and it is possible to to for instances to succeed even if the fidelity of entanglement is low (the requirement is $F > 0.5$) due to hardware imperfections.

To validate that the network schedules provided by Arqon support the successful execution of this application, the two end nodes in our network will repeatedly submit demands for the teleportation application and execute instances according to the network schedule provided. We will compare against a scenario in which the network schedules are simple time-division multiple access (TDMA) schedules with a fixed schedule period. These TDMA schedules were used in previous demonstrations of a two node network with programmable end nodes [22], and hence constitute the experimentally validated state of the art. The expected outcome of this test is that application instances will succeed with a higher probability when the Arqon schedules are used than when the TDMA schedules are used.

2. *Entanglement Purification*: In this application two entangled pairs between two nodes are successively produced with some initial fidelities F_1, F_2 to an ideal Bell-state. Gates are applied between the two qubits of each entangled pair at each node, and one of the qubits at each node is measured. In perfect operating conditions, the result is that the remaining entangled pair between the two nodes will have fidelity $F_3 > F_1, F_2$. Imperfections in the gates applied and additional errors accumulated by the first entangled entangled pair (resulting in fidelity $F_1^* < F_1$) during the time taken to produce the second entangled pair can result in a failure of this application. An instance of this application fails if the final entangled state has fidelity $F_3 <= F_1, F_2$ with respect to a Bell-state.

   This application requires the generation of a 2-pair packet of entanglement. We will execute the extreme-photon-loss (EPL) [33, 34] variant of this application, in which following production of the packet both nodes do a single 2-qubit gate (a controlled-not gate) between their local qubits and then measure their qubit belonging to the first entangled pair. This variant of entanglement purification includes less gates to perform on the entangled pairs (only 1 two-qubit gate per node), and hence is well-suited to scenarios where there is a high rate of error affecting the 2-pair packet of entanglement or the latency of applying gates is large compared to this rate of error accumulation.

   To validate Arqon's method of determining the amount of time to schedule for the generation of multiple pair packets (which is based on scan statistics) we will compare the expected probability of successful 2-pair packet generation with the observed probability of successful packet generation when this application is executed. In a more complex variant of this test, the two nodes will repeatedly submit demands for the EPL application, with subsequent demands specifying different values for the window of time in which the 2-pair packet must be generated. We

expect to observe a non-trivial trade-off between the duration of this window, the final fidelity F_3 of the entangled pair produced by a successful application instance, and the rate at which application instances succeed.

6.3.3. TEST PHASES

There are four phases to the integration testing:

1. **Preparation phase:** In the preparation phase upgrades are made to each subsystem (end node runtime, SDN controller, Arqon) and the interfaces between systems. The purpose of each upgrade is to support the successful systems integration.

2. **"Bare-metal" phase:** In this phase the quantum network applications included in the integration test plan are executed on the NV hardware of each node via a preprogrammed experimental sequence that directly actuates the control electronics of the QPS, bypassing the other elements of the Node Runtime. This phase establishes the baseline performance metrics of the application execution on these specific QPSs, in a controlled situation where no additional latency is introduced by classical processing or communication with the Node Runtime.

3. **"Dry-run" phase:** This phase tests the integration of Arqon, the SDN controller, and the end node runtimes, without the NV hardware. This phase includes testing the integration between the QPS and the CPS, but the NV hardware is disconnected from the rest of the QPS. The integration test applications are executed in the end node runtimes, which submit application demands to Arqon, wait for an acceptance response, and schedule entanglement generation needed for the application based on the network schedules produced by Arqon and delivered via the SDN controller. In this phase the NV hardware of each node remain disconnected from the end node runtimes, and are replaced by a mock system which simply responds to instructions from the end node runtime with acknowledgments of instruction receipt.

4. **Full-systems integration test phase:** This phase tests the full systems integration corresponding to Figue 6.3. The only difference from the dry-run phase is that the instructions sent to the QPSs by the end node runtimes now result in physical operations on the NV hardware, and quantum outcomes such as final state fidelity to an expected state can be extracted at the end of an application instance.

In these experiments we intend to demonstrate that the Arqon control architecture created in this thesis is suitable for operation of an entanglement-based network with imperfect hardware, and that it offers an improvement over minimal control strategies and hard-coded solutions. The integration process will enable the identification of features of the production version of Arqon which require adaptation in order to improve the efficacy of the total integrated system. As a result, an upgraded production version will be produced to support the end goal of deploying Arqon to control the QIA prototype network, illustrated in Figure 1.5 of the Introduction.

REFERENCES

[1] S. Gauthier, G. Vardoyan, and S. Wehner, *Method for controlling service demand of quantum entanglement generation in quantum network,* (2022).

[2] 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 **3**, 538 (2007).

[3] 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. **130**, 050803 (2023).

[4] H. K. Beukers, M. Pasini, H. Choi, D. Englund, R. Hanson, and J. Borregaard, *Remote-entanglement protocols for stationary qubits with photonic interfaces,* PRX Quantum **5** (2024), 10.1103/prxquantum.5.010202.

[5] 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 **132** (2024), 10.1103/physrevlett.132.150604.

[6] J. van Dam, G. Avis, T. B. Propp, F. Ferreira da Silva, J. A. Slater, T. E. Northup, and S. Wehner, *Hardware requirements for trapped-ion-based verifiable blind quantum computing with a measurement-only client,* Quantum Science and Technology **9**, 045031 (2024).

[7] A. J. Stolk, K. L. van der Enden, M.-C. Slater, I. te Raa-Derckx, P. Botma, J. van Rantwijk, J. J. B. Biemond, R. A. J. Hagen, R. W. Herfst, W. D. Koek, A. J. H. Meskers, R. Vollmer, E. J. van Zwet, M. Markham, A. M. Edmonds, J. F. Geus, F. Elsen, B. Jungbluth, C. Haefner, C. Tresp, J. Stuhler, S. Ritter, and R. Hanson, *Metropolitan-scale heralded entanglement of solid-state qubits,* Science Advances **10**, eadp6442 (2024).

[8] S. D. Barrett and P. Kok, *Efficient high-fidelity quantum computation using matter qubits and linear optics,* Physical Review A **71**, 060310 (2005).

[9] S. L. Braunstein and A. Mann, *Measurement of the Bell operator and quantum teleportation,* Physical Review A **51**, R1727 (1995).

[10] M. Michler, K. Mattle, H. Weinfurter, and A. Zeilinger, *Interferometric Bell-state analysis,* Physical Review A **53**, R1209 (1996).

[11] P. Walther and A. Zeilinger, *Experimental realization of a photonic Bell-state analyzer,* Physical Review A **72**, 010302 (2005).

[12] 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 **558**, 268 (2018).

[13] I. Pogorelov, T. Feldker, C. D. Marciniak, L. Postler, G. Jacob, O. Krieglsteiner, V. Podlesnic, M. Meth, V. Negnevitsky, M. Stadler, B. Höfer, C. Wächter, K. Lakhmanskiy, R. Blatt, P. Schindler, and T. Monz, *A compact ion-trap quantum computing demonstrator,* PRX Quantum **2**, 020343 (2021).

[14] A. Stolk, K. van der Enden, M.-C. Roehsner, A. Teepe, S. Faes, C. Bradley, S. Cadot, J. van Rantwijk, I. te Raa, R. Hagen, A. Verlaan, J. Biemond, A. Khorev, R. Vollmer, M. Markham, A. Edmonds, J. Morits, T. Taminiau, E. van Zwet, and R. Hanson, *Telecom-band quantum interference of frequency-converted photons from remote detuned nv centers,* PRX Quantum **3** (2022), 10.1103/prxquantum.3.020359.

[15] 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 **372**, 259 (2021).

[16] L.-M. Duan, M. D. Lukin, J. I. Cirac, and P. Zoller, *Long-distance quantum communication with atomic ensembles and linear optics,* Nature **414**, 413 (2001).

[17] A. Stolk, J. Biemond, K. van der Enden, L. van Dooren, E. van Zwet, and R. Hanson, *Extendable optical phase synchronization of remote and independent quantum network nodes over deployed fibers,* Phys. Rev. Appl. **23**, 014077 (2025).

[18] 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 **497**, 86 (2013).

[19] QuTech, *Quantum Network Explorer (QNE) – Unlocking the Power of Quantum Networks,* `https://www.quantum-network.com/` (2025).

[20] A. Gruber, A. Dräbenstedt, C. Tietz, L. Fleury, J. Wrachtrup, and C. von Borczyskowski, *Scanning confocal optical microscopy and magnetic resonance on single defect centers,* Science **276**, 2012 (1997).

[21] M. V. G. Dutt, L. Childress, L. Jiang, E. Togan, J. Maze, F. Jelezko, A. S. Zibrov, P. R. Hemmer, and M. D. Lukin, *Quantum register based on individual electronic and nuclear spin qubits in diamond,* Science **316**, 1312 (2007).

[22] C. Delle Donne, M. Iuliano, B. van der Vecht, G. M. Ferreira, H. Jirovská, T. J. W. van der Steenhoven, A. Dahlberg, M. Skrzypczyk, D. Fioretto, M. Teller, P. Filippov, A. R.-P. Montblanch, J. Fischer, H. B. van Ommen, N. Demetriou, D. Leichtle, L. Music, H. Ollivier, I. te Raa, W. Kozlowski, T. H. Taminiau, P. Pawełczak, T. E. Northup, R. Hanson, and S. Wehner, *An operating system for executing applications on quantum network nodes,* Nature **639**, 321–328 (2025).

[23] 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* (2019) pp. 159–173.

[24] 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 **8**, 121 (2022).

[25] M. Horodecki, P. Horodecki, and R. Horodecki, *General teleportation channel, singlet fraction, and quasidistillation,* Physical Review A **60**, 1888 (1999).

[26] C. Monroe, R. Raussendorf, A. Ruthven, K. R. Brown, P. Maunz, L.-M. Duan, and J. Kim, *Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects,* Phys. Rev. A **89**, 022317 (2014).

[27] J. I. Cirac, A. K. Ekert, S. F. Huelga, and C. Macchiavello, *Distributed quantum computation over noisy channels,* Physical Review A **59**, 4249–4254 (1999).

[28] P. Arright and L. Salvail, *Blind quantum computation,* International Journal of Quantum Information **04**, 883 (2006).

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

[30] D. Leichtle, L. Music, E. Kashefi, and H. Ollivier, *Verifying BQP computations on noisy devices with minimal overhead,* PRX Quantum **2**, 040302 (2021).

[31] H. Barnum, C. Crepeau, D. Gottesman, A. Smith, and A. Tapp, *Authentication of quantum messages,* in *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, SFCS-02 (IEEE Comput. Soc) p. 449–458.

[32] D. Unruh, *Quantum position verification in the random oracle model,* in *Advances in Cryptology – CRYPTO 2014*, edited by J. A. Garay and R. Gennaro (Springer Berlin Heidelberg, Berlin, Heidelberg, 2014) pp. 1–18.

[33] E. T. Campbell and S. C. Benjamin, *Measurement-based entanglement under conditions of extreme photon loss,* Physical Review Letters **101**, 130502 (2008).

[34] F. Rozpędek, T. Schiet, L. P. Thinh, D. Elkouss, A. C. Doherty, and S. Wehner, *Optimizing practical entanglement distillation,* Physical Review A **97**, 062333 (2018).

7

OUTLOOK

In this thesis we have developed control architectures for multi-user quantum networks. I concluded in Chapter 6 with a discussion of the next steps for deploying the developed architectures with real quantum network nodes. In the following section I focus on the centrally controlled architecture developed in Chapters 4 and 5 and discuss my outlook on protocols that still need to be developed to implement some of the features of this architecture. Implementing these features will become critical in deployed multi-user networks.

The architectures developed in this thesis all involve an element of central control. As the final discussion topic of this thesis I provide my outlook on extending beyond centrally controlled architectures, retaining a focus what I expect to be implementable with near-term quantum network nodes.

7.1. FURTHER PROTOCOL DEVELOPMENT FOR DEPLOYING MULTI-USER NETWORKS

7.1.1. PROTOCOLS IMPLEMENTING A NETWORK CAPABILITY UPDATE

The architecture developed in Chapters 4 and 5 requires a mechanism of learning the entanglement generation capabilities along end-to-end routes in the network. I envision that to learn these capabilities, the Network Manager could be able to submit a special type of demand to the Network Scheduler for the characterization of an entangled link or an end-to-end route. This characterization demand would be treated in the same way as other demands, with the difference that it is not associated with an application session on end nodes but rather with a characterization request of the Network Manager. Developing and testing such a mechanism for scheduling link or route characterization is a near term requirement to extend the rollout of our architecture beyond an experimental implementation. In an experimental implementation, this mechanism may be entirely circumvented by manually performing characterization of links between neighboring nodes or end-to-end routes before starting an experiment (in which the network schedule

is produced by Arqon). In this proposed experimental implementation, the network capability table is pre-populated by the entanglement generation capabilities and then not updated during the entire operation time of the experiment.

Once a mechanism is developed for scheduling the characterization of entanglement generation capabilities along links between neighboring nodes, it will also be beneficial to develop a (mathematical) protocol for establishing the end-to-end entanglement generation capabilities along a route from only this direct link characterization. The argument is that characterization of a link between neighboring nodes is expected to require much less time than characterization of an end-to-end route, as the entanglement generation rate along direct links will be much higher than for end-to-end routes with multiple hops. Therefore, it will be easier to schedule the characterization of direct links. This protocol should be validated against direct characterization of end-to-end routes, by measuring entanglement generation rate and fidelity along the route and comparing the values obtained experimentally and the values calculated by the protocol that takes only the direct link level capabilities as input. The protocol is useful if the experimentally derived values for a route are a close match (in terms of mean and standard deviation) to those that are calculated by the protocol.

7.1.2. Protocols Implementing Capability Negotiation

Before capability negotiation, pairs of end nodes must query the central controller for the entanglement generation capabilities on end-to-end routes. Then, the end nodes that will collaborate to execute an application must directly communicate to configure aspects of their local programs that depend on each nodes execution capabilities (such as a maximum number of qubits that can be allocated to the program or maximum quantum memory lifetimes), as well as on the end-to-end entanglement generation capabilities. Specific protocols need to be developed defining exactly which aspects of how a program will be executed are configurable and how a pair of nodes can reach a consensus on compatible configurations. Above, I gave examples of parameters that may be configurable, but a protocol must define these parameters precisely. I envision than such a protocol would be executed by the local runtime of an end node and would select between multiple distinct compilations of a program realizing one part of a quantum network application. The protocol would then configure the selected compiled program with any parameter settings specified as the outcome of the capability negotiation.

7.1.3. Configuring Application Demands

An end node must have a method of automatically configuring the demand for an application session based on the program that they will execute, the network capabilities, and capability negotiation with the other end nodes with which it collaborates to execute the application. One parameter which requires such configuration is the expiry time of a demand, which should be informed by the number of instances to execute, the expected duration of an instance conditioned on generating the required packets of entanglement, and the expected packet generation rate. Another parameter that must be configured is the window of time within which all entangled pairs in a packet should be generated. This window may be based on the memory lifetimes of specific qubits on the collaborating end nodes and on the expected duration of the subroutines within an application instance

that require the packet of entanglement.

7.1.4. ENTANGLEMENT GENERATION IN HETEROGENEOUS NETWORKS

In a heterogeneous network, different nodes will support different Remote Entanglement Generation Protocols (REGPs) [1, 2]. Concretely, consider the scenario where a node supports both detection in midpoint and sender in midpoint type REGPs, and it is connected to a metropolitan hub equipped with both a detection type Entanglement Generation Interface (EGI) (such as a Bell-state analyzer, see e.g. [3–5]) and a sending type EGI (such as an entangled photon source). This scenario has implications on physically motivated protocol development for both a network controller and the local control of end nodes. A network controller must have protocols for adapting routing to the physical modalities that an end node supports. An end node controller must have protocols for adaptively configuring the entanglement generation protocol to be executed based on the route selected to execute the entanglement generation.

7.2. BEYOND CENTRALLY CONTROLLED ARCHITECTURES

In Chapters 2 and 3 we developed control architectures for a stand-alone metropolitan hub. In contrast, in Chapters 4 and 5 we developed an architecture for a centrally controlled network where the logical controllers of components such as the metropolitan hub do not have autonomous control over scheduling or admission control for incoming demands. As we demonstrated in with the complexity analysis of the Network Scheduler in Chapter 5, the computation times of algorithms for routing, network scheduling, and admission control in a centrally controlled architecture can scale poorly with the number of resources in the network (in Chapter 5 the scaling is exponential with the number of resources).

An alternative to centrally controlled architectures are distributed architectures. In general, distributed architectures tend to demonstrate better scaling of algorithm computation times with the number of resources in the network, as compared with centrally controlled architectures [6, 7]. Centrally controlled architectures are also vulnerable to a situation in which communication between some or all nodes and the central controller is disrupted. In that situation, a centrally controlled network cannot function correctly. In contrast a distributed architecture may be able to function correctly, at least in some parts of the network, if communication is disrupted in some regions of the network [6, 7]. There is also a communication time barrier to scaling a centrally controlled network beyond a radius of hundreds or a few thousand kilometers. As the distance from nodes to a central controller increases, so to does the communication time of critical messages such as demand submissions, status notifications, and network schedules. A logically distributed architecture can mitigate delays as the radius of a network increases because control messages may only need to traverse shorter distances between nodes and local controllers, or directly between two nodes with autonomous control capabilities.

There are proposals for distributed quantum network control architectures, including [8, 9]. An outstanding challenge faced by a distributed architecture is to implement protocols ensuring the delivery of reliable service, in the sense developed in Chapter 5. The development of such protocols is more difficult in a logically distributed network than

in a centrally controlled network because there is no component with global knowledge of the current network traffic and the requirements to deliver reliable service to this traffic. In distributed classical networks, this is often overcome by over-provisioning the network, in the sense of deploying data links with a larger bandwidth than is required for serving the expected traffic over the the link [10]. In the over-provisioned scenario best-effort policies such as the TCP/IP protocol used in the classical internet can deliver reliable service [10].

In my opinion, the strategy of over-provisioning does not immediately extend to near-term quantum networks. In the quantum network context, over-provisioning links would require the ability to generate entanglement on significantly shorter timescales than the maximum entanglement storage time, and it would require nodes with a large number of quantum memories to store generated entanglement. To progress towards distributed architectures, while maintaining some of the benefits of a centrally controlled architecture, I propose developing a nested control architecture in which there is a central controller, but local controllers of components such as metropolitan hubs have increased autonomy. In such an architecture demands could be submitted by end nodes to local controllers, such as the controller of a metropolitan hub, and only forwarded from the hub to a higher level controller, such as a central controller if the demand is a non-local demand, meaning it does not correspond to two end nodes in the same metropolitan network. The architectures developed in Chapters 2 and 3 could form the basis for the semi-autonomous control of metropolitan hubs, and the centrally controlled architecture developed in Chapters 4 and 5 could form the basis for the treatment of non-local demands.

7

REFERENCES

[1] H. K. Beukers, M. Pasini, H. Choi, D. Englund, R. Hanson, and J. Borregaard, *Remote-entanglement protocols for stationary qubits with photonic interfaces,* PRX Quantum **5** (2024), 10.1103/prxquantum.5.010202.

[2] C. Jones, D. Kim, M. T. Rakher, P. G. Kwiat, and T. D. Ladd, *Design and analysis of communication protocols for quantum repeater networks,* New Journal of Physics **18**, 083015 (2016).

[3] S. L. Braunstein and A. Mann, *Measurement of the Bell operator and quantum teleportation,* Physical Review A **51**, R1727 (1995).

[4] M. Michler, K. Mattle, H. Weinfurter, and A. Zeilinger, *Interferometric Bell-state analysis,* Physical Review A **53**, R1209 (1996).

[5] P. Walther and A. Zeilinger, *Experimental realization of a photonic Bell-state analyzer,* Physical Review A **72**, 010302 (2005).

[6] P. Baran, *On distributed communications networks,* IEEE Transactions on Communications Systems **12**, 1 (1964).

[7] M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed. (distributed-systems.net, 2017).

[8] L. Bacciottini, M. G. D. Andrade, S. Pouryousef, E. A. V. Milligen, A. Chandra, N. K. Panigrahy, N. S. V. Rao, G. Vardoyan, and D. Towsley, *Leveraging internet principles to build a quantum network,* IEEE Network , 1 (2025).

[9] 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)* (2022) pp. 341–352.

[10] A. S. Tanenbaum, *Computer Networks*, 4th ed. (Prentice Hall PTR, Upper Saddle River, New Jersey, 2003) p. 912.

ACKNOWLEDGMENTS

Stephanie, from our earliest introduction I learned that you had an ambitious plan for driving the development of an entirely new technology. At the time you sent me a draft of *A Vision for the Road Ahead* and encouraged me to apply for an MSc scholarship if I was interested in the roadmap you and Ronald and David defined there. Over the years of working together since you have been and continue to be an important role model for me. I am inspired by the way you spend time and energy on a crafting clear and ambitious vision and use that to drive the identification of what work needs to be done to make that vision a reality. When you agree to work with someone, you provide a strong sense that you are confident in their abilities and that you believe that they are equal to the task before them. Thank you for that trust, for me it was exactly what I needed in order to grow into a successful researcher. I appreciate that I have been able to speak openly and honestly with you throughout the years on my strengths, my weaknesses, and my ambitions. As we continue to work together, I still find it valuable to learn from how you approach different challenges and opportunities.

Ronald, thank you for being my co-promoter, it has been very helpful to discuss solid state quantum network node systems with you and members of your group over the last years. I valued the opportunity to easily—sometimes by simply walking downstairs—ask questions about your group's results and experience in implementing these systems. What I learned from these interactions helped me to tailor design considerations so thar they may correspond to real quantum network nodes. I enjoyed the opportunity to collaborate on an invention together with **Ingmar**, **Arian**, and **Mariagrazia**, it was great to learn about the interesting systems results that can emerge by bringing together a multi-faceted group with experience in hardware, experiments, software, and theory.

To the members of my committee **Yaroslav Blanter**, **Fernando Kuipers**, **Michele Amoretti**, **Rihan Hai**, and **Anna Pappa**, I would like to offer my heartfelt gratitude for taking the time to read and challenge my dissertation. I consider it a great privledge to have the opportunity to discuss my work with each of you.

Thomas thank you for being such an excellent collaborator. Working with you has made almost every aspect of my PhD experience better, more efficient, and more enjoyable. I found our way of working very refreshing. It felt like no matter the technical problem we were facing, we were eventually able to solve it just by talking in front of a white board for too many hours in a row and remembering to write down our conclusions. After one such meeting, I remember the time we laughed until tears from stumbling onto a fake wiki article written as if by a time traveler and covering the false history of the IP addressing system up to something like IPv12. Over the course of our PhDs we had the opportunity to travel together far and wide for conferences and QIA events. As travel companions during the local explorations that followed some of these trips I thought we gelled very well on our interest for outdoor explorations and willingness to step off the beaten path. One of my favorite moments happened in Utah during the US road trip,

when you and **Luca** and I had failed to win the Angel's Landing lottery for the following day. I really wasn't expecting the unbridled enthusiasm that met my suggestion of reserving canyoning gear and spending the following day wading through sometimes chest deep water in a slot canyon. In the near term at least, it seems our post-PhD roles will lead us to collaborate occasionally. I look forward to continuing to discuss networking and system integration challenges and opportunities.

Janice, you and I had very similar parallel PhD experiences, where beyond the typical research we were both also tasked with leadership roles for important work efforts within QIA. Seeing as in the beginning, neither of us had any idea what we each felt that we were completely learning on the job in these roles, I am really glad to have had each other to look to and laugh about the experience with. As my closest female friend within the group over the years, thank you for being there and for letting me come and go freely from your office whenever I felt like a small chat or pick-me-up. I appreciate that you are so kind and energetic and generally easy to be around, never introducing unnecessary complications into the situation. I also appreciate that you are not scared to disagree with people or to hold your ground on defending your opinions or discussion positions. Thank you for the sunny memories and nice breakfast hangouts over the years. I especially appreciated spending time together on our most recent QIA work visit to Innsbruck, it was really fun to spend time chatting about everything and nothing on the couch of our Airbnb before bed.

Andrei and Sasha, The best way I can express the role you have both played in my journey from start to end of the PhD is that you have both become a part of the bedrock of my life in Delft, it's foundation. No matter what I was encountered over these years, you both made me feel like I had people I could trust, people who made me feel welcome, and people who make me feel more at home in the world. I appreciate your fresh points of view and ever-questioning natures. Countless times you have helped me to bolster my confidence in the face of challenges, both in everyday life and in those encountered on adventures. Thank you for being true friends and I appreciate so much all of our long coffee chats, thought experiments, and climbing try-hard sessions. Most of all I appreciate looking back on all the shared moments enjoying time in the sun, or under the shade of a large rock feature.

Hana, Thank you for calling me from the side of the road in Spain following a smashed window and a change of plans. For me that moment was crucial in our friendship. Thank you for bringing me my passport in Italy, there aren't many people I would trust to handle that particular document. We'll never know if it would have been fine either way, but because of you we never have to find out. From before I first met you, I already had a positive opinion of you, as a clever computer scientist who clearly brought joy to those around her and had the respect of older colleagues. On getting to know you, my best suspicions were confirmed. I learned that you are both an amazing outdoor partner, both to explore and to relax with, as well as an astute and highly organized collaborator. This year I have learned that you are ferocious in games and that I will never never be able to get a round goal from you as easy pickings. I respect you so much for being brave enough to face difficult conversations and situations head on and with emotional honesty. Thank you for all the acts of kindness and care over the years, you have absolutely made these years better and brighter.

Frida, although technically we were introduced once beforehand, I like to think that we met in France when I semi-crashed your vacation and climbing trip. For the first three years of my PhD, the idea of being in Delft without you also there fighting the good fight was unimaginable. Thank you for laughing with me about administrative absurdities and bureaucratic horrors. Those cathartic moments have certainly given me the courage to conquer them. You're such a beautiful person and you bring such a uniquely insightful way of looking at things that is as valuable in your research and work as it is in your personal relationships. I have always felt that our values and priorities are so well aligned that they hardly needed any explanation, yet it was always wonderful to hear each others view on the nuances. Since you moved on to starting your career in Oslo, I'm really happy to see you enjoying and living life to the fullest in a place where you can be a little closer to nature. I'm looking forward to the next time we escape somewhere together.

Tadas, I really appreciate the way you operate as a friend. Always kindly accepting that what I see as a departure time, you see as an arrival time; making us both coffee while I get started on dinner/breakfast/getting ready to leave; agreeing to an successfully planning climbing trips with no hesitation as soon as we propose a date. I really enjoyed training with you and since your recovery from injury also our occasional bouldering sessions, even though you sometimes had to convince me to come to the Hague. Over the years my most unhinged and contagious fits of laughter have been triggered with you. Thank you for trusting me and letting me in as a close friend, I always feel I can tell you the truth and I'm really grateful that you have always checked in and arranged meetups periodically, even in times when I was not very on top of my personal communication.

Alex, I have always considered you to be a role model in some ways, as you are so amazingly organized, brilliant, kind, and incredibly hard-working. You are by far the most goal-oriented person I have ever met, and it's exceedingly clear that once you have set a goal—whether it be athletic, career, personal, academic—there is nothing on this earth that can stand in your way. I really value being in each other's lives and I am glad that we have been able to see each other all across Canada and even in France over the years since I moved to Delft. I often think back on our rainy day hike in Vancouver or our day of good but spontaneous decision making while biking/hiking and eventually ice cream eating in Lyon. I'm glad that I will always have a reminder of that day.

Marnix, I'm not really sure why but our personalities have always clicked such that we end up laughing about something completely ridiculous almost every time we meet. Thank you for all of those laughs, for your friendship, and for always having time for a good chat in the gym. Your work ethic, especially as applied to training, is truly inspiring.

Zuza, Rocco, Phil, and Zohar, thank you for the shared adventures both in Delft and running around the mountains. Each of you helped open my eyes to new experiences, and I wouldn't be the same person I am today without those moments.

Gayane Vardoyan, you were an excellent mentor to me during the first years of my PhD. In the final days of working out the results for our on-demand paper, it was a rush to chase out the remaining mismatches between numerical evaluation of the model and simulation results. I learned a lot from you about how to approach preparing a paper for submission. Your attention to detail always impresses me. I often think of you when double checking every claim in preparing a proof or calculation.

Wojciech Kozlowski I learned a lot about fundamental principles of network architec-

tures and protocol design from discussions early in my PhD. Quite early on you helped me to understand that the network architectures I would develop in my PhD needed to be practical to integrate and operate in the context of the QIA network. These systems level considerations ended up being what was constantly in the forefront of my mind when working on Arqon. On a personal level I appreciate your direct, open and accepting manner. I like that you question everything and always seem to draw your own conclusions.

Tracy Northup, Ben Lanyon, and **Fokko de Vries**, each of you have been incredibly helpful and wonderful to collaborate with as QIA members over the last several years. I would like to thank each of you individually for taking the time to meet, discuss and answer my questions about each of your hardware systems. Each of you helped me to state realistic requirements for metropolitan hubs, and taught me about physical or technical considerations that have shaped design considerations in all of my other work. I look forward to continuing to work together in QIA now and in the future.

Thirupathaiah Vasantam, I enjoyed collaborating on our on-demand paper. I learned a lot from you about queuing theory during our research discussions. I enjoyed working out as a team, together with **Gayane Vardoyan**, which service models were both practical for operation of the hub system as well as physically relevant for realistic quantum devices.

Arian Stolk thank you for so many knowledge sharing discussions over the years about NV systems. I also enjoyed collaborating on our shared patent, and on exploratory technical discussions for network protocols. While you were at QuTech, it was always fun to accidentally sink in to a lengthy political discussion, whether in the hallway, at a defence party, or a QuTech barbeque. I really enjoyed both ramen and pizza dinners with you, Vincy and Kian over the last few years. I hope you and Vincy are off to a great start in Singapore.

Mariagrazia Iuliano, I enjoyed collaborating with you on our invention, learning from you about network experiments with NV, and exploratory technical discussions for network protocol design. Personally, I think you are an impressive scientist—smart, organized, a leader, stylish, and fun to be around. I hope you are enjoying your new role in Munich.

Nina Condreanu, over the years I have enjoyed getting to know you through nice coffee table conversations at weird times of day. Since the first time I saw you give a WB I have thought you are a great scientist and excellent at communicating your deep knowledge of materials science and fabrication techniques. Now that we are both in our new post PhD roles, I'm happy we can still have coffee table chats at odd hours.

Michal van Hooft, thank you for helping my MSc students with NetSquid features, with setting up hub scheduling architectures using NetSquid Netbuilder, and for thinking along in the early days of QI Lab on whether NetSquid would be suitable for adaption to that real time use case. I've enjoyed getting to know each other over the last years and it's nice to see you more frequently since I've moved to the E wing.

Nico Seidler, thank you for your gentle encouragement in all matters QIA over the last years. Early on in my PhD I felt a little bit clueless in the SET team meetings, but your way of leading the team made me feel welcome. Towards the end of my PhD, thank you for the role you played in helping to make the Arqon integration experiments using QNE hardware a reality. I am glad to continue working together through QIA.

Thom van der Steenhoven, it was great to work together on the design of interfaces between Arqon and QNodeOS and later on the actual software integration tests. Throughout, your sharp wit and sense of humor really shone through, making all of our efforts more enjoyable. To you, **Bart van der Vecht**, **Thomas Beauchamp**, **João Pedro Marques Rôla**, and **Pieter Botma**, a million thank yous for your patience and diligence in debugging the integrated system.

Bart van der Vecht, thank you for sharing your insights on local program execution and scheduling in discussions over the years. They were really helpful in designing our network architecture and implementation to be compatible with local node operating systems. You were really a leader in creating the QNodeOS features for supporting the Arqon integration, and towards the major crunch time I was impressed with the speed at which you were able to deliver new features and documentation.

Pieter Botma, I really enjoyed collaborating with you on the Arqon integration experiments. Throughout the entire process the Arqon team benefited from your deep knowledge of the full QNE hardware setup. The long back and forth exchanges via teams over the best way to structure entanglement generation sessions in order to comply with PGAs were super appreciated.

Ingmar te Raa-Derckx, over the years I am glad that we have successfully collaborated in different ways, including our shared invention with members of the Hanson lab, and on the systems integration of Arqon with QNodeOS. I appreciate that you can always describe a system clearly and logically. It's always nice to see you around QuTech or at QIA events.

Vlora Rexhepi-van der Pol, I admire your calm and clear way of speaking and giving presentations. During your time working with QIA you were a role model for me of how to carry oneself in professional settings. I remember several times when you encouraged me in collaboration settings, and those moments were meaningful to me and helped to grow my confidence.

Karolina Zykovic, thank you for all your help with scheduling my defense and ensuring a smooth transition for me between my PhD and starting in my new role with QIA. I really appreciate your sardonic humor and hope to laugh together when new mini-crises arise in the future.

Marit Talsma, as my first MSc student, thank you for trusting me to supervise your thesis and for all of the energy you brought to working on the project. Your confident, direct, and resourceful personality brought a special energy to the group.

Christiaan Korbee, from the beginning of your MSc project, including in our first discussion of the project description, you always asked excellent and insightful questions. It was fun to work with you on your MSc project. I thought it was fantastic that you always independently proposed solutions accompanied by extensive analysis, and collaboratively asked for my input to check assumptions and discuss conclusions and next steps. I really admire your organization and determination, and the concise way you solve problems.

Jake Smith, I'm really happy to work together and have been very impressed by your organization, drive, and leadership already early on in your PhD. On our collaboration visit to Innsbruck this year together with **Sam**, **Oumayma**, and **Janice** earlier this year I was pleasantly surprised to learn about your deadpan comic delivery, which on that trip sparked a lot of laughter for our group. Waiting for the text from Felix when time was

passing and nobody believed you that he would still text was truly hilarious, and it was awesome that meeting up worked out in the end.

Milen Kolev, I'm very glad to have had the opportunity to work with you on QI lab. In taking on the project, you quickly transformed it from a nice idea into a functional research tool, and you are constantly improving it. I think you have great stories and have really enjoyed exchanging anecdotes from outdoor sports stories, travel, and wider views about the world and technology.

Sacha Bernheim, I enjoyed our occasional evening visits to the route climbing gym and the time we visited the Kletterzentrum, together with **Sam**, in Innsbruck following the QIA Spring Meeting, let's do it again sometime! I had a fun evening celebrating with you and **Milen**, and **Luca** after submitting my thesis and finishing my PhD contract.

Sam Oslovich, it's been great collaborating with you on the QIA test plans, I appreciate how you take complete ownership of your part of any task. I always laugh when I think of the occasions since you joined the group in which you and I, and now also **Jake** have formed a sort of North American coalition where we collectively regard something as normal (which it apparently is not) or in which we recognize a concept with which our European compatriots are unfamiliar.

To the entire systems team of the Wehner group, **Anders Rehult, Jake, Luca, Milen, Oumayma, Sacha, Sam, Thomas,** thank you for the rich discussions and teamwork, I look forward to continuing to collaborate on the system development topics pertaining to QIA.

Luca Marchese, I had a great time hiking and exploring with you on our road trip together with **Thomas** following the March meeting. I was so happy that both of you were on board with the plan of camping and climbing at Joshua tree. The whole experience—including cooking over the fire with far too few utensils—was one of my favorite parts of the trip. I have to say that I was a little scared the entire time we were in the lava cave in Arizona, but it was a very unique hike and I was also happy we could do something that you were really excited about. It's always fun to discuss physics with you, your deep understanding really shows through. Thank you again for reading the proofs for Arqon, your feedback was super helpful.

Jeroen Grimbergen, over the years I have really enjoyed arguing (also known as debating) topics with you over lunch, coffee, or at a QINC beach day. It was fun to explore a little bit of LA together during our March meeting week. Thank you for using your cash to buy the strawberries from the night market.

Juliette van Mil, I've enjoyed learning about your work from lunch and coffee chats, it seems to me like your PhD is off to a great start. I think you have the best outfits—I love that your looks are always complete, down to the details of your shoes and accessories. Thank you for helping me with taking a head shot, as you may have noticed it's getting some good use.

Soubhadra Maiti, thank you for sharing your knowledge on the different border node and interface scenarios for the prototype network. Best wishes for the final part of your PhD.

Kaushik Senthoor, thank you for giving me feedback on an earlier version of my thesis introduction. Your insistence on rigor is what finally convinced me I needed to spend more time on quantum states. During your postdoc in the Wehner group I enjoyed our sporadic discussions about novels. I hope you are doing well in your new position in the

UK.

Tzula Propp, I have enjoyed conversations with you about foundations of quantum mechanics and appreciate your exuberant approach to your work. You are always both kind and encouraging to me and everyone else in your path, I hope you are doing well in Leiden.

Oumayma Bouchmal, we had a brief but glorious run with an all female office. I always enjoy our conversations, especially about just being a girl in the world with dreams and trying to balance having a personal life and having a lot of work to do. I look forward to the next stage of organizing the Lorentz center workshop together, and to further collaboration on the control plane optimizations for QIA.

Sabrina Álvarez Vásquez, I enjoyed our brief run as office mates this year following the start of your PhD. You seem to have a bright outlook and a methodical approach to problem solving on the white board. With all your questions, you strike me as quite a curious person. I think you will do well, and I wish you the best with your PhD.

Sergio Loarte, I have enjoyed hearing some of your stories from previous work in industry and living in different places. I wish you success with the rest of your PhD.

Margaret Atwood and **Mary Pratt**, thank you for sharing your astute powers of observation with the world. Your works have inspired me continuously to share my own observations with the world.

Uncle Roger, during my PhD your enthusiasm for my research and my future career has been so genuine that it has honestly helped to make me feel like my work can have an impact in the wider world. During my undergrad, the weekends and Easters we spent together at your place in Toronto really made me feel that I had a home there. You have a very special place in my heart, thank you for always encouraging me and for adding wonderful sprinkles of whimsy to my life. The small enamel cardinal and engraved loon sit proudly on my desk and bookshelf, respectively.

Grand-maman and Papa, your love and encouragement means the world to me. I thought it was very special that you surprised at Joanne's cottage during my last visit to Ontario. Thank you both for cheering me on.

Aunt Joanne and Uncle Peter, from the very first days of my life in Delft you two were amazingly helpful and supportive. Thank you so much for all your efforts to get me set up and comfortable after making such a big change. Throughout all stages of my academic journey your support has helped me immensely, enabling me to start my professional life off on the right track. Over the years of my PhD I have had a lot of fun visiting you at your cottage in Canada, or more recently spending time together exploring Amsterdam in ways that I had never considered before. Thank you both for all your care and kindness.

Marijke and Peter, thank you for making space for me in your little family. I really appreciate the kindness you have shown me and the encouragement you have given me whenever I have explained my current workload or upcoming objectives that were causing some stress. Peter, I always look back very fondly on our skiing trip with Kian to St. Anton. It was a wonderful week and I really felt that experience made us grow closer. Marijke, thank you for sharing your love of coffee chats and LKKR!!! with me. On summer nights I fondly remember you Kian and I—sometimes Peter too— rushing to finish dinner and run out the door in order to make it in time for ijs (met slagroom) on a sunny evening. Thank you both for small acts of kindness and support over the last years and especially

since Kian has been away, like taking me to Schiphol or bringing by a portion of tomato soup.

Dad and Danielle, I appreciate all of the encouragement you have given me over the years that I have been in Delft, thank you for visiting me here a couple of times. In the first year of my PhD I really appreciate you including me in the grand opening of your Astana based lonely hearts hotel for daughters going through a breakup. I had a great time exploring Kazakhstan with you on that trip, thank you for making all of my wishes happen. Dad, thank you for always telling me to have fun first and only reminding me to be careful second. Danielle, thank you for sharing your emotional intelligence with us, I really appreciate it.

Sky, you may be my only sister, but you are hands down the best sister. You are so good at slipping in creative one liners to a conversation, you make me laugh every time I see you. Your constant positive and encouraging outlook on my life always makes me feel like maybe I am doing something right. Thank you for visiting me in Delft multiple times over these years and meeting up with me in various Canadian cities. I always love talking to you—knowing that you simply get it makes me feel that life is just a little bit easier.

Kian, you have been on this journey with me through thick and thin. Thank you for listening to me, strategizing with me, helping me believe I can successfully navigate bureaucracy, and talking to me late into the night—both about dreams in life as well as how to stop stressing and start dreaming. On a professional level, it's been very helpful to be able to ask you NV related questions over the years, thank you for entertaining work related discussions in unlikely moments walking to the grocery store or cooking dinner. I always love discussing science or business with you, but we have never had any shortage of topics to discuss otherwise either. I smile when I think about all the moments in which a chat ran far overtime in hallways and at coffee tables. The constant of having weekend pancake breakfast to look forward to was a cozy, grounding routine that took the edge off of busy weeks and felt like hitting the reset button. I look forward to exploring together in all the years to come. Together strong, I know we can build the life of our dreams. Just in case a reminder is needed, please don't forget to bring the rope.

Scarlett
Vancouver, August 2026

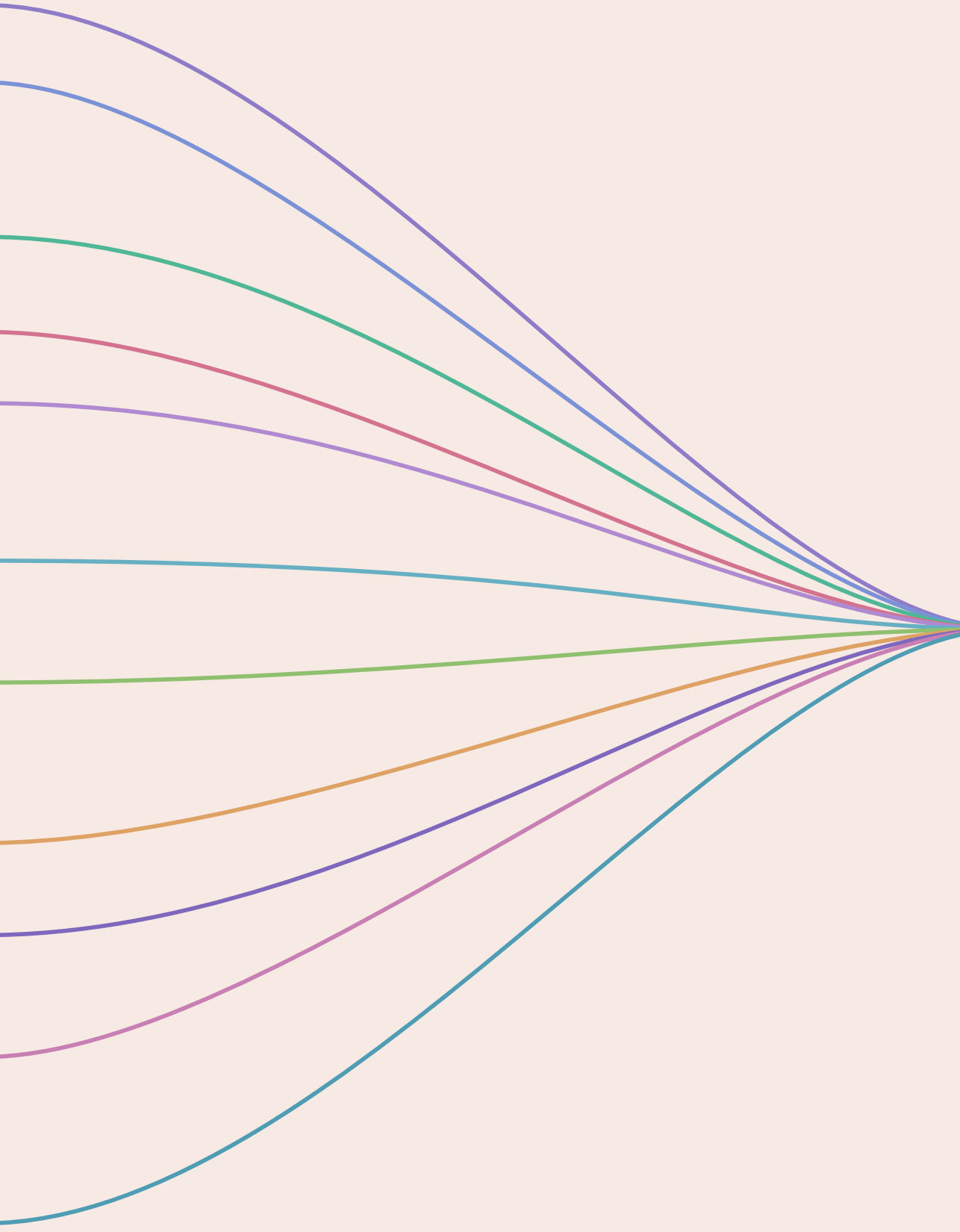
CURRICULUM VITÆ

Scarlett Savannah GAUTHIER
December 12th, Ottawa, Canada

Sept.-Dec. 2016	**Math Center Resource Person** Humber Polytechnic, Toronto, Canada
May-Aug. 2017	**Research Assistant**, CUTE: Cryogenic Underground TEst facility SNOLAB Underground Science Facility, Sudbury, Canada
2017	**First place, Summer Student Talk Competition** SNOLAB Underground Science Facility, Sudbury, Canada
Jan-Aug. 2018	**Research Assistant**, Loncar Group Harvard University, Boston, The United States
2014-2019	**Bachelor of Science**, Mathematical Physics, Distinction *Honours* University of Waterloo, Waterloo, Canada Thesis: *"Strain-Tuning Based Stabilization of Silicon Vacancy Color Centers in Diamond"* Supervised by: Prof. dr. Marco Loncar (Harvard University) Awarded a *President's Entrance Scholarship*

(Continued on next page)

2020	**First place, Quantum Future Vision Contest** For the article *Projective Measurements: tracing out the impact of quantum technology in 2040*, co-authored with Brennan Undseth Delft University of Technology, Delft, The Netherlands
2020-2021	**Teaching Assistant** Delft University of Technology, Delft, The Netherlands For the courses *Quantum Communication and Cryptography (MSc Level)* and *Mathematics for Quantum Physics (BSc Level)*
April-July 2021	**Simulation Engineer, Intern**, Lattice Surgery for Surface Codes HQS Quantum Simulations, Karlsruhe, Germany
2019-2021	**Master of Science**, Applied Physics, Distinction *Cum Laude* Delft University of Technology, Delft, The Netherlands Thesis: *"Recurrence Based Purification of Qudit Graph States"* Supervised by: Prof. dr. Stephanie Wehner Awarded a *QuTech Scholarship* Completed the *Quantum Technology Degree Annotation*
2024-2026	**Technical Advisor**, Architecture Team The Quantum Internet Alliance (QIA), Delft, The Netherlands
2021-2026	**Ph.D, Computer Science and Applied Mathematics** QuTech, Delft University of Technology, Delft, The Netherlands Thesis: *"Control Architectures for Multi-User Quantum Networks"* Promotors: Prof. dr. Stephanie Wehner, Prof. dr. ir. Ronald Hanson
April 2026—	**Lead Mission Engineer**, Moonshot Mission to Build a Prototype Quantum Network QIA, QuTech, Delft University of Technology, The Netherlands



ISBN 978-94-6563-003-8