Software Abstractions for Programmable Quantum Network Nodes

Carlo Delle Donne

Software Abstractions for Programmable Quantum Network Nodes

Software Abstractions for Programmable Quantum Network Nodes

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. T. H. J. J. van der Hagen,
chair of the Board for Doctorates,
to be defended publicly on
Tuesday, 27 June 2023 at 12:30 o'clock

by

Carlo DELLE DONNE

Master of Science in Embedded Systems,
Delft University of Technology, the Netherlands,
born in Potenza, Italy.

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof. dr. S. D. C. Wehner,	promotor
Dr. P. Pawełczak,	promotor
Independent members:	
Prof. dr. A. van Deursen,	Delft University of Technology
Dr. L. Y. Chen,	Delft University of Technology
Prof. dr. R. Van Meter,	Keio University, Japan
Dr. C. G. Almudéver,	Technical University of Valencia, Spain
Prof. dr. K. G. Langendoen,	Delft University of Technology, reserve member

Keywords:	Quantum networks, operating systems
Cover:	無音の暴君 (*Silent tyrant*), Mari Nakagawa
Style:	TU Delft House Style, with modifications by Moritz Beller `https://github.com/Inventitech/phd-thesis-template`
Printed by:	Ipskamp Printing `https://www.ipskampprinting.nl/proefschriften/`

An electronic version of this dissertation is available at
`https://repository.tudelft.nl/`.

ISBN 978-94-6384-457-4

It’s tempting to linger in this moment,
while every possibility still exists.
But unless they are collapsed by an observer,
they will never be more than possibilities.

Solanum

Contents

Summary

Computer networks have been one of the most revolutionary concepts and technologies of the last fifty years. Currently, it is arguably impossible to imagine a world without the internet. And yet, just five decades ago, hardly anybody knew what it even meant. Today, the first quantum computer networks are starting to take shape, along with the promise of a future quantum internet. Quantum networking exploits fundamental primitives of quantum mechanics — most importantly *entanglement* — to offer a new paradigm of connectivity, which will enhance communication networks and bring some new exciting applications into the scene.

Quantum networking has been studied for a few years already. Nevertheless, the current state of the art of quantum networks is somewhat comparable to that of the classical internet at the end of the 1960s: lots of interesting ideas, some experimental demonstrations, and very few reliable testbeds. Scaling up to larger networks of quantum computers requires joint efforts of physics, mathematics, electronics and computer science, at the very least. Bringing these disciplines together is a very bumpy road, given that we do not yet have standard quantum physical platforms to work with, nor universal frameworks and testbeds to validate our hypotheses against. One of the missing links between the highly-complex physical platforms and networks and the high-level descriptions of quantum networking applications is a framework that bridges that gap between these two, providing platform-independent abstractions of the underlying physics to programmers and users of a quantum network.

The goal of this thesis is threefold: discuss the requirements for such a framework of abstractions — which we refer to as an operating system — for quantum networks, propose a design for such an operating system, and implement and validate this design on a physical quantum network. Whilst we are interested in measuring the performance of the operating system, we consider our design to be best-effort, and thus we are primarily aiming at establishing a baseline for future research in this field. Nevertheless, we are after a fully-functional product that we hope can be used to push the boundaries of quantum networking demonstrations, and to better understand the challenges of designing and implementing efficient operating systems for quantum network nodes.

Samenvatting

Computer netwerken is een van de meest revolutionaire concepten and technologieën van de afgelopen vijftig jaar. Tegenwoordig is het zo goed als onmogelijk om je een wereld zonder het internet voor te stellen. En toch, slechts vijf decennia geleden, wist bijna niemand wat het ook maar betekende. Nu wordt gewerkt aan de eerste kwantum computer netwerken, met daarbij een belofte op een toekomstig kwantum internet. Kwantum netwerken maken gebruik van de fundamentele beginselen van de kwantummechanica — waarbij vooral het concept van *verstrengeling* van belang is — om een nieuw paradigma van connectiviteit aan ge bieden, welke communicatie netwerken zal versterken en nieuwe, spannende toepassingen zal doen opbrengen.

Kwantum netwerken worden al een aantal jaar bestudeerd. Desondanks is de status van kwantum netwerken tegenwoordig ongeveer vergelijkbaar met dat van het klassieke internet aan het eind van de jaren 60: veel interessante ideeën, een aantal experimentele demonstraties, en weinig betrouwbare testbeds. Opschalen naar grotere netwerken met kwantum computers heeft op zijn minst een collectieve inspanning van natuurkundigen, wiskundigen, elektrotechnici en informatici nodig. Deze disciplines samen laten komen is een lastige taak, aangezien we nog geen standaard fysieke platformen hebben, nog hebben we universele kaders en testbeds om onze hypothese mee te testen. Eén van de ontbrekende schakels tussen complexe fysieke platformen en netwerken en de abstracte omschrijvingen van kwantum netwerk toepassingen is een kader dat het gat daartussen overbrugt door platform-onafhankelijke abstracties van de onderliggende natuurkunde aan te bieden aan programmeurs en gebruikers van het kwantum netwerk.

Het doel van deze scriptie heeft drie hoofdzaken: de benodigdheden van een dergelijk kader van abstracties voor kwantum netwerken — welke we een besturingssysteem noemen — bediscussiëren, een ontwerp voor zo'n besturingssysteem aandragen, en dit ontwerpen implementeren en valideren op een fysiek kwantum netwerk. Waar we wel geïnteresseerd zijn in de prestaties van het besturingssysteem, noemen we ons ontwerp "best-effort", and richten we ons vooral op het neerzetten van een uitgangspunt voor toekomstig onderzoek in dit onderzoeksgebied. Desondanks zijn we op zoek naar een volledig functionerend product waarvan we hopen dat het de grenzen van kwantum netwerk demonstraties kan verleggen, en om een beter begrip te krijgen van de uitdagingen in het efficiënt implementeren van besturingssystemen voor kwantum netwerk nodes.

Acknowledgments

Whilst this thesis bears my name as the only author, all of this work and my whole personal and professional development would not have been remotely possible without the help of a number of people. There is no way a few paragraphs can express my gratitude satisfactorily, but I want you to know that your support has been invaluable and essential.

My two promotors Stephanie and Przemek have been my guiding light throughout this four-year scientific journey. Ronald has provided a cutting-edge testbed to enable groundbreaking experiments. The committee members Arie, Carmina, Lydia and Rodney have provided solid feedback to help me refine this dissertation and have accepted to wear Medieval heat-trapping robes at the end of June to attend my defence. My fellow group mates Álvaro, Bart, Bethany, Francisco, Guus, Hana, Janice, Ravi, Scarlett, Soubhadra and Thomas, and other present and past QuTechers Alejandro, Aram, Axel, David, David, Eric, Gayane, Guilherme, Helena, Ingmar, Josh, Joël, Kanchan, Kenneth, Luise, Mariagrazia, Matt, Matt, Matteo, Nico, Önder, Paul, Ravi, Remon, Siddhant, Sébastian, Thom, Thomas, Tim, Vicky and Wojtek have been the pillar of my work and life at QuTech. My friends from Italy Alessio, Davide, Davide, Edoardo, Eleonora, Francesco, Giulia, Manuela, Pierluigi, Rosita and Simone, and those found in the Netherlands Costas, Dimitris and Patrick have never failed to make me feel loved and never cease to being a massive source of joy, energy and inspiration. My whole family, and particularly Mamma, Papà, Luigi and my companion Giulia are my shell, my den, my all-inclusive five-star suite, they are me, and I am them.

Carlo
Delft, June 2023

Curriculum Vitæ

Carlo Delle Donne

2018 – 2023	Ph.D. Quantum Computer Science Delft University of Technology, The Netherlands Thesis: *Software Abstractions for Programmable Quantum Network Nodes* Promotors: Prof. dr. S. D. C. Wehner, Dr. P. Pawełczak
2016 – 2018	M.Sc. Embedded Systems Delft University of Technology, The Netherlands Thesis: *Wake-up Alignment for Batteryless Sensors* Advisor: Dr. P. Pawełczak
2013 – 2016	B.Sc. Electronics and Telecommunications University of Bologna, Italy Thesis: *Real-Time Data Streaming Using Bluetooth Low Energy* Advisors: Dr. E. Farella, Dr. B. Milosevic, Dr. S. Benatti
1994/05/02	Born in Potenza, Italy

List of Publications

6. **C. Delle Donne**, M. Iuliano, B. van der Vecht, M. Skrzypczyk, I. te Raa, G. M. Ferreira, T. van der Steenhoven, A. R.-P. Montblanch, M. Pompili, S. L. N. Hermans, N. Demetriou, B. van Ommen, T. H. Taminiau, P. Pawełczak, W. Kozlowski, R. Hanson, and S. Wehner. "QNodeOS: An Operating System for Quantum Network Nodes"

   This article is included in this thesis as Chapters 2, 3 and 5.
   This article is in preparation.

5. J. S. Abrahams, **C. Delle Donne**, P. Brussee, T. Middelburg, R. C. Berrevoets, J. A. Slater, and S. Wehner. "Data Origin Authentication in the Quantum Network Protocol Stack"

   J. S. Abrahams and **C. Delle Donne** contributed equally to this article.
   This article is included in this thesis as Chapter 6.
   This article is in preparation.

4. M. Pompili, **C. Delle Donne**, I. te Raa, B. van der Vecht, M. Skrzypczyk, G. M. 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". In: *npj Quantum Information* 8.1 (2022), p. 121. DOI: 10.1038/s41534-022-00631-2

   M. Pompili and **C. Delle Donne** contributed equally to this article.
   This article is included in this thesis as Chapter 4.

3. A. Dahlberg, B. van der Vecht, **C. Delle Donne**, M. Skrzypczyk, I. te Raa, W. Kozlowski, and S. Wehner. "NetQASM—A Low-Level Instruction Set Architecture for Hybrid Quantum–Classical Programs in a Quantum Internet". In: *Quantum Science and Technology* 7.3 (2022), p. 035023. DOI: 10.1088/2058-9565/ac753f

2. J. de Winkel, **C. Delle Donne**, K. S. Yıldırım, P. Pawełczak, and J. Hester. "Reliable Timekeeping for Intermittent Computing". In: *ASPLOS*. ACM, 2020, pp. 53–67. DOI: 10.1145/3373376.3378464

1. A. Y. Majid, **C. Delle Donne**, K. Maeng, A. Colin, K. S. Yıldırım, B. Lucia, and P. Pawełczak. "Dynamic Task-Based Intermittent Execution for Energy-Harvesting Devices". In: *ACM Trans. Sen. Netw.* 16.1 (2020), pp. 1–24. DOI: 10.1145/3360285

1 Introduction

Quite possibly, the word "quantum" is one of the most trending of the last decade, primarily in the context of science and technology, but not only. Attaching it to the name of a product makes it sound more advanced, superior. Leveraging people's confusion over the meaning of quantum mechanics, the *q*-word has been (mis)used to advertise, among other things, computer applications [41], cleaning products [40], and pseudoscience-fueled health and medical advice [22].

However, quantum physics is not just an otherworldly theory from science fiction books, nor just a catchy name for 21st-century consumer products. Since the formulation of the theory of quantum mechanics in the early decades of the 20th century, researchers and enthusiasts have been looking into how to make use of these physical properties, particularly in the fields of electronics, information processing and telecommunications. Integrated circuits, lasers, light-emitting diodes and magnetic resonance imaging devices are just a few example technologies that rely on the quantum theory to function. In the domain of computer science, we know of a handful of applications that exploit the axioms of quantum information theory to achieve something that was though to be very hard, or even impossible. Some of the most well-known use cases include fast resolution of computational problems — like integer factorization [28] and unstructured searching [12] — and efficient and secure communication schemes — for instance quantum key distribution [4, 10] and superdense coding [3].

Quantum networking, a new paradigm of telecommunications, seeks to enhance — not replace — our current internet technology to provide new functionalities that are impossible to attain with purely classical communications. Novel applications include security-enhancing communication schemes such as quantum key distribution (QKD) [4, 10], advanced clock synchronization routines [15], distributed consensus protocols [2], distributed sensing [11], and secure cloud quantum computing [5, 6]. Even though quantum communications are an established reality, and their potential applications have garnered attention from industry and research institutes, the above list of applications does not necessarily appeal to the masses, which mostly feel puzzled when someone tries to pitch their research on quantum networking. Nonetheless, the community of quantum networking researchers is not discouraged by this mismatch in expectations, as it hopes that more applications will be devised once the technology becomes more widespread and available to more "consumers". After all, we were also not aware of all the possible uses of the classical internet when it was first developed. Yet, today the internet means instantaneous access to low-cost clothes, scenes of hilarious felines, and — for some — tapes of bare bodies engaging in intimate action on camera.

One of the most frequently asked questions about quantum technology is: *When can we use it?* If more people had access to quantum networks, they would perhaps come up with more ideas for useful quantum networking applications. Thus, what is missing before we can deploy quantum networks consisting of a useful number of nodes? What are the main limitations we are facing? How can we overcome them? Not surprisingly, the answers to these questions are complicated. There is a cauldron of fundamental and theoretical limitations, technological hardware obstacles, and computer science puzzles that hinder the success of quantum networking. Fortunately, though, there is a growing community of passionate researchers trying to study these limitations, build better hardware, and solve these puzzles. In this thesis, we will address some of the challenges arising when trying to manage the resources and the activity of a quantum network node, from an operating system's perspective. We thus aim to answer a fraction of the research questions in the field of quantum networking, and to lay another brick in the construction of *scalable*, *controllable*, and *configurable* quantum networks.

The remainder of this chapter walks the reader through basic networking concepts and quantum networking challenges, lists the research questions we aim to address, and provides an outline of the rest of the thesis.

1.1 Networking and Quantum Networking

Digital communication networks have come a long way from the early days of the ARPANET — one of the most important precursors of today's internet. Back in 1969, one of the most advanced networks of computers consisted of just *four* nodes. In 1981, as the global network grew larger, networking researchers standardized the Internet Protocol version 4 (IPv4), which allowed up to 4 billion devices to have their own address on the public internet [36]. Soon after, it became clear that the pool of available IPv4 addresses was going to be depleted sooner than later — which happened in 2011 [38]. In 2023, there are 3.6 devices connected to the internet per capita, as estimated by Cisco in 2020 [39].

Scaling up from a four-node experiment to the massive networks of the 21st century was no easy feat of course. This was made possible by advancements in various fields, including networking hardware, traffic engineering, and network programmability. You would most likely not be able to download a digital copy of this thesis in a fraction of a second if it were not for Tbit/s network switches [42, 43], a diverse spectrum of routing protocols [34, 35, 37], and software-defined networking [18], among other things. Nevertheless, classical networking was already appealing in its infant stages, for the simple reason that even a small network of nodes can accomplish tasks that would not be attainable without it — in the case of the ARPANET, sending simple pieces of text over large distances almost instantaneously. The applications of quantum networking are not dissimilar to their classical networking counterpart, in that some of them can be useful and effective on small quantum networks already, whilst other use cases require more powerful nodes and more complex networks [31].

Even though hardware and software advancements were essential to the betterment of the internet, there are a few basic design ingredients that have been there since the dawn of classical networking and that have made these technological leaps even possible. Key architectural principles that guided the design of computer networks include the *end-to-end principle* [27], *encapsulation* [33], and the *packet switching model* [26]. When designing quantum networks and networking services, one should draw inspiration from classical networking principles and avoid monolithic designs and software architectures that would render the system rigid and hard to scale.

Then, how similar are quantum networks and classical networks? Which are the challenges that they have in common, and which ones are exclusive to quantum networking? How much can we capitalize on the vast body of classical networking literature? In principle, quantum networks are just networks with a special physical layer, which, albeit more technologically complex, could be abstracted away and encapsulated into an ad-hoc networking protocol. This simplification disregards, however, some fundamental limitations that are inherent to quantum information, as well as the imperfect nature of near-term quantum hardware. Standard networking routines like signal amplification, classical error correction, and data retransmission would not work in quantum networks — one fundamental theorem of quantum mechanics states that *an arbitrary quantum state cannot be cloned* [9, 32]. Moreover, quantum states are subject to *decoherence*, a physical process whereby the quality of the stored information degrades over time and due to external interferences. Thus, not only do computation and networking delays affect throughput and latency, but they also exact a toll on the quality of the service, effectively determining whether a certain application produced meaningful results or not. Whilst decoherence can be worked around with more sophisticated quantum hardware and control algorithms, the no-cloning theorem is a hard limit on what one can do with quantum information, and thus we cannot design quantum networking protocols assuming quantum information can be freely copied and stored indefinitely.

Without being too speculative, we can argue that we cannot just encapsulate the requirements of quantum networking into a specialized physical layer. Reusing classical networking techniques and protocols as they are would fall short of the aforementioned challenges posed by quantum mechanics. Nonetheless, many of the questions that drove the classical networking research community can be of inspiration for analyzing requirements and limitations of quantum networks. Examples of such questions are: *Can we organize quantum information into packets? Do we need to resort to path reservation for quantum communications? In which situations can we tolerate local and network latency? How do we organize and layer quantum networking protocols and services? What metrics do we look at when designing routing protocols? Can we improve performance and quality of service with the employment of a software-defined control plane?*

1.2 Research Goals

Perhaps disappointingly, but unsurprisingly too, this thesis will not try to answer all the research questions from the previous section. The good news is that there are many ongoing efforts from various scientists that are looking into these questions, particularly focusing on networking protocols [7, 14, 16, 23, 29], SDN [1, 17], network architectures [8, 13, 19, 20, 21, 25], and node architectures [30], among other topics. Most of the times, however, researchers have to validate their designs on a simulated quantum network, given that we do not yet have access to mid-scale testbeds consisting of more than a handful of nodes — the most advanced of which features three interconnected devices [24]. On the other hand, we need a framework to evaluate quantum networking protocols and applications on real quantum networks, to help us verify our assumptions and simulation results even in the early stages of this research field. In this thesis, we will discuss design considerations for such a framework, design and implement a rudimentary instance of it, and evaluate our design and implementation on a quantum network.

The goal of this thesis is to provide a framework that facilitates the experimental investigation of quantum networking-related research questions, and that can help researchers learn about the behavior of quantum communication applications without having to delve into the complexity of running and managing the underlying network and the interaction of the nodes with it. We refer to this framework as an *operating system* (OS), as its goal is to abstract and manage quantum physical processes and resources to provide a user-friendly interface to the application. More specifically, we will address the following questions:

Q1. *What goals should an OS for quantum network nodes achieve?* We explore challenges, requirements and goals that one should consider when designing such an OS, whose overarching objective is to bridge the gap between high-level user applications and low-level quantum networking hardware.

Q2. *What does an architecture for such an OS look like?* We propose the first proof-of-principle architecture for an OS for quantum network nodes. The architecture ensures the OS can be deployed on various quantum platforms, provides means to manage resources and schedule operations, and allows running multiple quantum networking applications concurrently.

Q3. *What is the performance of the OS's quantum network stack?* We revise and implement state-of-the-art quantum network protocols to be integrated in the proposed OS, and evaluate their performance for a basic networking feature: entanglement delivery.

Q4. *What is the performance of the whole OS?* We implement the proposed architecture, and design test cases aimed at evaluating its functioning and performance, including basic quantum networking applications and scenarios of concurrent execution of multiple applications.

Q5. *How would data origin authentication affect the performance of a quantum link?* We evaluate, this time in simulation, what penalty would be incurred in the rate of entanglement generation if the quantum network stack would communicate over an authenticated classical channel, as opposed to exchanging non-authenticated classical messages. The results from this evaluation will guide the adoption of data origin authentication schemes into future versions of the proposed OS.

When measuring performance, we are mostly interested in two types of indicators: (1) *User-level quantum metrics* — most importantly quantum *fidelity*, which is typically an indicator of the quality of execution of the application, and which depends on the behavior of the OS as well as the quality of the quantum hardware. (2) *OS-level timings* — classical metrics like latency and throughput at various levels of the OS that provide insights into bottlenecks and shortcomings of the design and/or implementation of the OS itself, which

Object	**Location**
Datasets for Chapter 4	DOI: `10.4121/16912522`
Datasets for Chapter 6	DOI: `10.4121/d2726121-354d-444c-b859-3585d62811e5`
Application SDK	`https://github.com/QuTech-Delft/netqasm`

Table 1.1: Location of experimental data and software supporting this thesis.

in most cases have an impact on the quantum fidelity.

This work is aimed at designing, implementing and evaluating a "product" that, although experimental and not production-ready, is a fully-functional research tool, and as such is ready to be reused and adapted with little effort by anyone who is interested in experimenting with quantum networking protocols and applications on real networks. Our implementation-driven research does not intend to produce the best protocols and algorithms for the control of quantum network nodes — rather, it wishes to establish a baseline for such a system, and a framework to study and test more advanced versions of its components. To demonstrate the applicability of our tool, we evaluate the OS on a small state-of-the-art quantum network based on nitrogen-vacancy centers in diamond [24], deployed in a laboratory environment.

1.3 Data and Software Availability

The datasets that support this thesis are made public and available in the online data repository 4TU.ResearchData. The software to analyze such data is also made public and available in the same package as the datasets. The application software development kit (SDK) is open-sourced on GitHub. Finally, the operating system is a software project owned and developed by QuTech and the Delft University of Technology. As of the time of writing the project is not open source and not available for general usage. If you are interested in accessing the source code for research purposes, please contact QuTech at `secr-qutech@tudelft.nl`. Refer to Table 1.1 for pointers to data and software.

1.4 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 offers some background on quantum information, quantum networking and classical networking, and recaps the main challenges involved. Chapter 3 outlines the most important design considerations that serve as the basis for the design of our OS (question Q1), and describes our proposal for the architecture of a quantum network node's OS (question Q2). Chapter 4 showcases the performance of the quantum network stack integrated in the OS for entanglement generation (question Q3). Chapter 5 described the implementation of the OS and proposes test cases for evaluating its performance, including simple quantum networking applications and concurrent execution of multiple applications (question Q4). Chapter 6 quantifies, this time in simulation, the effect of authenticating classical messages in the quantum network stack on the performance of a quantum link (question Q5). Finally, Chapter 7 concludes this thesis and reflects upon future steps.

References

[1] A. Aguado, V. López, J. P. Brito, A. Pastor, D. R. López, and V. Martin. “Enabling Quantum Key Distribution Networks via Software-Defined Networking”. In: *ONDM*. IEEE, 2020, pp. 1–5. DOI: 10.23919/ONDM48393.2020.9133024.

[2] M. Ben-Or and A. Hassidim. “Fast Quantum Byzantine Agreement”. In: *STOC*. ACM, 2005, pp. 481–485. DOI: 10.1145/1060590.1060662.

[3] C. H. Bennett and S. J. Wiesner. “Communication via One- and Two-Particle Operators on Einstein-Podolsky-Rosen States”. In: *Physical Rev. Lett.* 69.20 (1992), pp. 2881–2884. DOI: 10.1103/PhysRevLett.69.2881.

[4] C. H. Bennett and G. Brassard. “Quantum Cryptography: Public Key distribution and Coin Tossing”. In: *Theor. Comput. Sci.* 560.1 (2014), pp. 7–11. DOI: 10.1016/j.tcs.2014.05.025.

[5] A. Broadbent, J. Fitzsimons, and E. Kashefi. “Universal Blind Quantum Computation”. In: *FOCS*. IEEE, 2009, pp. 517–526. DOI: 10.1109/FOCS.2009.36.

[6] A. M. Childs. “Secure Assisted Quantum Computation”. In: *Quantum Inf. Comput.* 5.6 (2005), pp. 456–466. DOI: 10.26421/QIC5.6-4.

[7] A. Dahlberg, M. Skrzypczyk, T. Coopmans, L. Wubben, F. Rozpędek, M. Pompili, A. Stolk, P. Pawełczak, R. Knegjens, J. de Oliveira Filho, R. Hanson, and S. Wehner. “A Link Layer Protocol for Quantum Networks”. In: *SIGCOMM*. ACM, 2019, pp. 159–173. DOI: 10.1145/3341302.3342070.

[8] S. DiAdamo, B. Qi, G. Miller, R. Kompella, and A. Shabani. “Packet Switching in Quantum Networks: A Path to the Quantum Internet”. In: *Phys. Rev. Res.* 4.4 (2022), p. 043064. DOI: 10.1103/PhysRevResearch.4.043064.

[9] D. Dieks. “Communication by EPR Devices”. In: *Physics Letters A* 92.6 (1982), pp. 271–272. DOI: 10.1016/0375-9601(82)90084-6.

[10] A. K. Ekert. “Quantum Cryptography Based on Bell’s Theorem”. In: *Phys. Rev. Lett.* 67.6 (1991), pp. 661–663. DOI: 10.1103/PhysRevLett.67.661.

[11] D. Gottesman, T. Jennewein, and S. Croke. “Longer-Baseline Telescopes Using Quantum Repeaters”. In: *Phys. Rev. Lett.* 109.7 (2012), pp. 070503-1–070503-5. DOI: 10.1103/PhysRevLett.109.070503.

[12] L. K. Grover. “A Fast Quantum Mechanical Algorithm for Database Search”. In: *STOC*. ACM, 1996, pp. 212–219. DOI: 10.1145/237814.237866.

[13] H. Gu, Z. Li, R. Yu, X. Wang, F. Zhou, and J. Liu. “FENDI: High-Fidelity Entanglement Distribution in the Quantum Internet”. 2023. arXiv: 2301.08269.

[14] J. Illiano, M. Caleffi, A. Manzalini, and A. S. Cacciapuoti. “Quantum Internet Protocol Stack: a Comprehensive Survey”. 2022. arXiv: 2202.10894.

[15] P. Kómár, E. M. Kessler, M. Bishof, L. Jiang, A. S. Sørensen, J. Ye, and M. D. Lukin. “A Quantum Network of Clocks”. In: *Nature Phys.* 10.8 (2014), pp. 582–587. DOI: 10.1038/nphys3000.

[16] W. Kozlowski, A. Dahlberg, and S. Wehner. “Designing a Quantum Network Protocol”. In: *CoNEXT*. ACM, 2020, pp. 1–16. DOI: 10.1145/3386367.3431293.

[17] W. Kozlowski, F. Kuipers, and S. Wehner. "A P4 Data Plane for the Quantum Internet". In: *EuroP4*. ACM, 2020, pp. 49–51. URL: 10.1145/3426744.3431321.

[18] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. "Software-Defined Networking: A Comprehensive Survey". In: *Proceedings of the IEEE* 103.1 (2014), pp. 14–76. DOI: 10.1109/JPROC.2014.2371999.

[19] Z. Li, K. Xue, J. Li, N. Yu, D. S. L. Wei, and R. Li. "Connection-Oriented and Connectionless Remote Entanglement Distribution Strategies in Quantum Networks". In: *IEEE Network* 36.6 (2022), pp. 150–156. DOI: 10.1109/MNET.107.2100483.

[20] R. Mandil, S. DiAdamo, B. Qi, and A. Shabani. "Quantum Key Distribution in a Packet-Switched Network". 2023. arXiv: 2302.14005.

[21] T. Matsuo, C. Durand, and R. Van Meter. "Quantum Link Bootstrapping Using a RuleSet-Based Communication Protocol". In: *Phys. Rev. A* 100.5 (2019), p. 052320. DOI: 10.1103/PhysRevA.100.052320.

[22] L. R. Milgrom. "Patient-Practitioner-Remedy (PPR) Entanglement. Part 1: A Qualitative, Non-local Metaphor for Homeopathy Based on Quantum Theory". In: *Homeopathy* 91.04 (2002), pp. 239–248. DOI: 10.1054/homp.2002.0055.

[23] A. Pirker and W. Dür. "A Quantum Network Stack and Protocols for Reliable Entanglement-Based Networks". In: *New Journal of Physics* 21.3 (2019), p. 033003. URL: 10.1088/1367-2630/ab05f7.

[24] 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". In: *Science* 372.6539 (2021), pp. 259–264. DOI: 10.1126/science.abg1919.

[25] S. Pouryousef, N. K. Panigrahy, and D. Towsley. "A Quantum Overlay Network for Efficient Entanglement Distribution". 2022. arXiv: 2212.01694.

[26] L. G. Roberts. "The Evolution of Packet Switching". In: *Proceedings of the IEEE* 66.11 (1978), pp. 1307–1313. DOI: 10.1109/PROC.1978.11141.

[27] J. H. Saltzer, D. P. Reed, and D. D. Clark. "End-To-End Arguments in System Design". In: *ACM Transactions on Computer Systems (TOCS)* 2.4 (1984), pp. 277–288. DOI: 10.1145/357401.357402.

[28] P. W. Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: *FOCS*. IEEE, 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[29] R. Van Meter and J. Touch. "Designing Quantum Repeater Networks". In: *IEEE Communications Magazine* 51.8 (2013), pp. 64–71. DOI: 10.1109/MCOM.2013.6576340.

[30] G. Vardoyan, M. Skrzypczyk, and S. Wehner. "On the Quantum Performance Evaluation of two Distributed Quantum Architectures". In: *Performance Evaluation* 153 (2022), pp. 1–26. DOI: 10.1016/j.peva.2021.102242.

[31] S. Wehner, D. Elkouss, and R. Hanson. "Quantum Internet: A Vision for the Road Ahead". In: *Science* 362.6412 (2018), pp. 1–9. DOI: 10.1126/science.aam9288.

[32] W. K. Wootters and W. H. Zurek. “A Single Quantum Cannot Be Cloned”. In: *Nature* 299 (1982), pp. 802–803. DOI: `10.1038/299802a0`.

[33] R. T. Braden. *Requirements for Internet Hosts – Communication Layers*. RFC 1122. 1989. URL: `https://datatracker.ietf.org/doc/html/rfc1122`.

[34] International Organization for Standardization. *Intermediate System to Intermediate System Intra-Domain Routeing Information Exchange Protocol for Use in Conjunction With the Protocol for Providing the Connectionless-Mode Network Service (ISO 8473)*. ISO/IEC 10589:2002. 2002. URL: `https://www.iso.org/standard/30932.html`.

[35] J. Moy. *OSPF Version 2*. RFC 2328. 1998. URL: `https://datatracker.ietf.org/doc/html/rfc2328`.

[36] J. Postel. *Internet Protocol*. RFC 791. 1981. URL: `https://datatracker.ietf.org/doc/html/rfc791`.

[37] Y. Rekhter, S. Hares, and T. Li. *A Border Gateway Protocol 4 (BGP-4)*. RFC 4271. 2006. URL: `https://datatracker.ietf.org/doc/html/rfc4271`.

[38] *Available Pool of Unallocated IPv4 Internet Addresses Now Completely Emptied*. ICANN. URL: `https://itp.cdn.icann.org/en/files/announcements/release-03feb11-en.pdf` (visited on Feb. 28, 2023).

[39] *Cisco Annual Internet Report (2018–2023) White Paper*. Cisco. URL: `https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html` (visited on Feb. 28, 2023).

[40] *Finish Quantum Dishwasher Tablets*. Finish. URL: `https://www.finishdishwashing.com/products/detergents/quantum-detergent/` (visited on Feb. 28, 2023).

[41] *Introducing the New Firefox: Firefox Quantum*. Mozilla. URL: `https://blog.mozilla.org/en/mozilla/introducing-firefox-quantum/` (visited on Feb. 28, 2023).

[42] *QFX5220 Switches*. Juniper. URL: `https://www.juniper.net/us/en/products/switches/qfx-series/qfx5220-data-center-switches.html` (visited on Feb. 28, 2023).

[43] *Tomahawk 5 Ethernet Switch Series*. Broadcom. URL: `https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm78900-series` (visited on Feb. 28, 2023).

2
Quantum Networking: Background and Challenges

Quantum networks are a fundamentally new paradigm of telecommunications, which promise to enhance our classical networking primitives to achieve unprecedented tasks in various areas of communications and sensing. But how do quantum networks differ from their classical counterpart? What makes them special, and at the same time challenging to manage? This chapter provides useful background knowledge on quantum networking and recaps the primary challenges thereof.

This chapter is extracted from the article in preparation: **C. Delle Donne**, M. Iuliano, B. van der Vecht, M. Skrzypczyk, I. te Raa, G. M. Ferreira, T. van der Steenhoven, A. R.-P. Montblanch, M. Pompili, S. L. N. Hermans, N. Demetriou, B. van Ommen, T. H. Taminiau, P. Pawełczak, W. Kozlowski, R. Hanson, and S. Wehner. "QNodeOS: An Operating System for Quantum Network Nodes".

Contributions: S. Wehner conceived the project. **C. Delle Donne**, B. van der Vecht, W. Kozlowski, M. Skrzypczyk, I. te Raa, P. Pawełczak and S. Wehner analyzed the challenges and designed the OS. **C. Delle Donne**, B. van der Vecht, W. Kozlowski, I. te Raa, G. M. Ferreira, T. van der Steenhoven, M. Skrzypczyk and P. Pawełczak implemented the OS. M. Iuliano, A. R.-P. Montblanch, M. Pompili, S. L. N. Hermans, N. Demetriou and B. van Ommen prepared the quantum physical layer apparatus. **C. Delle Donne**, M. Iuliano, B. van der Vecht, M. Pompili, S. L. N. Hermans, W. Kozlowski, I. te Raa, R. Hanson and S. Wehner devised the experiments. **C. Delle Donne**, W. Kozlowski, P. Pawełczak and S. Wehner wrote the manuscript with input from all authors. T. H. Taminiau, R. Hanson and S. Wehner supervised the research.

Ubiquitous internet connectivity has already unlocked a plethora of applications that were not even conceived just years ago. Similarly, a future quantum internet [22, 44] aims to connect quantum devices — the *end nodes* — over large distances, in order provide new internet functionality that is impossible to achieve using solely classical communication. Examples of applications running on end nodes include security-enhancing protocols such as quantum key distribution (QKD) [5, 16], improved clock synchronization [23], support for distributed sensing [18] and distributed systems [4], as well as secure quantum computing in the cloud [11, 12].

To run a general quantum networking application, the end nodes' hardware-software system needs to be capable of performing certain actions, as summarized in Figure 2.1. First, nodes must be able to establish a quantum connection by generating quantum *entanglement* between them. Entanglement is a special property of at least two quantum bits — or *qubits* — one held by each end node. The entangled qubits are often measured directly by the application, or may be used to transmit data qubits from one end node to the other through teleportation [6]. Alongside entanglement, end nodes must be capable of executing local quantum operations on the qubits held by an end node, that is, quantum *gates* and quantum *measurements*. For simple quantum applications such as secure communication [5, 16] it is sufficient to produce entanglement and then perform a local measurement at each end node. However, for more complex quantum applications — enabled at higher stages of quantum internet development [44] — local operations can include the execution of quantum gates and in fact full quantum computation on a quantum processor. Finally, next to such quantum actions, most quantum applications known to date require local classical processing, as well as classical communication between the end nodes.

Abstractly, a quantum networking application consists of multiple programs, each running on one of the end nodes. The distinct programs only interact with one another by means of entanglement generation and classical communication. This allows a programmer to realize security-sensitive applications just as in the classical domain, but prohibits a global orchestration of the quantum execution as one might do in quantum computing. The case of secure quantum computing in the cloud [11, 12] is an example of a quantum networking application, schematically depicted in Figure 2.2. In blind quantum computing, a client node wants to perform a computation on a remote server node, the latter being

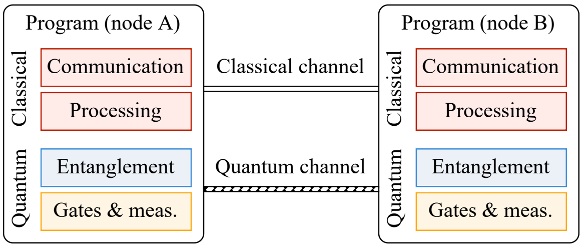


Figure 2.1: A quantum networking application consists of separate programs running at two or more end nodes that communicate via classical message passing and quantum entanglement. Local operations include quantum operations (gates and measurements) as well as classical processing.

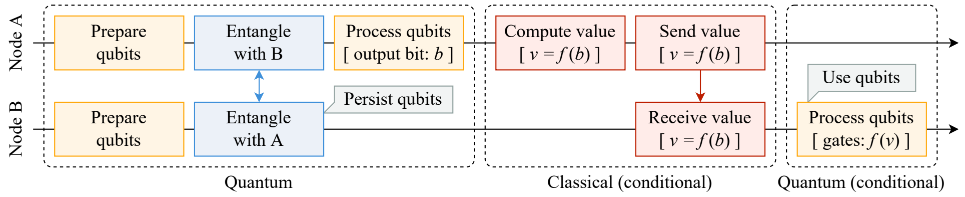

Node A
Node B
Prepare qubits
Entangle with B
Process qubits [output bit: b]
Prepare qubits
Entangle with A
Persist qubits
Compute value [$v = f(b)$]
Send value [$v = f(b)$]
Receive value [$v = f(b)$]
Use qubits
Process qubits [gates: $f(v)$]
Quantum
Classical (conditional)
Quantum (conditional)

Figure 2.2: Structure of a typical quantum network application (blind quantum computation [11, 12]), which consists of interleaved quantum processing blocks and classical processing blocks. Quantum processing blocks include local quantum operations (gates and measurements, yellow boxes) and network operations (entanglement generation, blue boxes). The execution of some classical and quantum blocks might be conditional on classical and quantum data coming from previous blocks. Qubit states in quantum blocks may have to persist ("Persist qubits") to be used in later quantum blocks ("Use qubits"), e.g. following the reception of classical messages from the remote node.

a powerful quantum computer, without the server learning anything about the computation. Blind quantum computing illustrates the need for a continuing interaction between the classical and quantum parts of the execution, such as waiting for a message from a remote client before continuing the quantum execution at the server. It also highlights the need for both classical and quantum state to be kept alive, for example such that future quantum instructions can be executed depending on messages from remote end nodes. This is in sharp contrast to quantum computing applications, where one can process the entire quantum execution in a single batch.

Up to now, demonstrations of quantum networking beyond QKD focused on hardware realizations. Different types of end node quantum hardware have been realized, ranging from simple photonic devices on which the only operation is a measurement [39, 45], to fully-fledged quantum processors with a network interface [7, 20, 29, 34, 37]. The largest quantum network linking quantum processors to date connects three nodes [34] based on nitrogen-vacancy centers in diamond (NV centers) at the physical layer. Demonstrations of applications beyond QKD have been performed using several photonic devices [3, 9, 31, 38]. Central to all these demonstrations is that the software to control the hardware was specific to the experiment setup, written to perform one single task (the experiment itself) and programmed into low-level control devices. In fact, often applications were not even actually fully realized towards a user, and instead they were meant to show that the hardware is in principle good enough for that specific application [26, 46].

In order to advance quantum networks from a physics experiment to fully-fledged systems, we need a combined software-hardware system that is built as a series of abstraction layers. These abstractions should expose a simple interface for the user to write applications in high-level, platform-independent software, and be able to interact with a variety of candidate platforms for future quantum network hardware. When designing such a system, many challenges arise (refer to Section 2.2 for details), which can be roughly classified into three areas. First, there exist *fundamental differences* between classical and quantum communication. A example of this is the concept of heralded entanglement generation, which requires coordinated actions by both nodes involved — that is, the operations to produce entanglement need to be scheduled at both nodes at the same time. Second, the *technological limitations* of near-term quantum devices impose stringent demands on the performance of such a system. One example is that the same quantum device is used for processing as well as networking, which implies that local operations cannot be scheduled independently of network operations — which in turn depend on the remote node. Finally, we remark that, unlike in the study of classical operating systems, which take advantage of the existence of advanced computer architectures defining a specific interaction of software and hardware, there exists *no general low-level quantum processor architecture*. Ideally, our system should be able to operate under the stringent constraints imposed by current technological limitations, but should also not be tailored to near-term quantum devices only.

2.1 Background

Here, we define some basic concepts of quantum networking hardware and performance metrics, necessary to understand the remainder of this chapter and this thesis. We refer the reader to the book *Quantum Computation and Quantum Information* by Nielsen and Chuang [32] for a general introduction to quantum information, and to the article "A Link Layer Protocol for Quantum Networks" by Dahlberg et al. [13] for more information on quantum networking.

Quantum network nodes. Generally speaking, a quantum network node is a quantum processor (or device) with an optical interface for external communication (entanglement). The processor can perform operations on one or more qubits. Local quantum operations range from simple qubit measurements [39, 45] to universal quantum computation [7, 20, 29, 34, 37]. Quantum networking operations allow certain qubits to produce entanglement with a remote node. In practice, only some types of qubits are suited for entanglement generation — we refer to such qubits as *communication qubits* — and only these qubits have an optical interface to the outside world. Other types of qubits — referred to as *storage qubits* — are instead more suited for storing quantum states for longer times (up to seconds in some cases [1, 10]). Often, storage qubits can also be used to process quantum information directly. On some quantum devices instead, for instance nitrogen-vacancy centers in diamond (NV centers), communication qubits are also the main gateway for local quantum gates, meaning that most processing operations need to step though these qubits, and thus their usage needs to be shared between local quantum computation and entanglement generation. What operations can be performed on what types of qubits depends on the specific quantum device. We remark that, at this stage of technological development, qubits, as well as any quantum operations applied on them, are not perfect. In fact, their quality even depends on the specific device sample being used.

Timing constraints. Quantum devices are generally controlled using a variety of classical signal generators, depending on the quantum device itself. For instance, in NV centers, the quantum device is controlled using microwave as well as laser pulses. The device-level control must satisfy hard real-time constraints and timing precision — nanosecond precision with sub-nanosecond jitter — and is realized using waveform generators, lasers, and custom electronics assisted by a dedicated microcontroller. Entanglement generation between two nodes connected by an optical fiber also requires the same scale of timing synchronization between the two devices [13, 35]. The low-level control of other quantum platforms is realized similarly [29, 37]. On top of those constraints, qubits have a limited lifetime — the states they hold must be processed before they become invalid. On some devices, qubit lifetimes have been shown to exceed one second [1]. Nevertheless, the quality of a qubit state is not constant throughout its lifetime, but it *decoheres* (becomes worse) at a certain rate. Qubit lifetimes and decoherence, however, are technological limitations, rather than fundamental ones, and are expected to become more tractable in the future.

Performance metrics. Next to standard classical performance metrics such as latency and throughput, the performance of quantum networking applications hinges on the quality of the quantum execution too. In the quantum networking domain, it is not generally an objective to eliminate all errors towards the application level [13, 43], and hence the performance of any operating system for quantum network nodes would be measured by the execution quality, and by the trade-offs with classical performance metrics [13, 43]. This quantum quality is generally measured by the quantum *fidelity* $F \in [0, 1]$, where a higher value corresponds to higher quality. For a quantum state, F measures the quality with respect to an ideal state. For a quantum gate or measurement, it measures the quality of execution, averaged over all possible states that it could be applied to. For a specific application, F can be translated into its quantum performance.

2.2 Challenges

Whilst the high-level goals for an operating system for quantum nodes mimic those of a classical operating system, we face a number of general challenges that are inherent to (near-term) quantum network nodes and network applications. Some of the challenges are technological limitations of the quantum physical layer, and we expect them to be less relevant in the future with further progress in quantum hardware development. To a certain extent, these issues apply to quantum computing too, from which we can draw some inspiration for their solutions. These challenges include:

1. limited available qubits, which imposes strict limits on the processing, networking, and storage capabilities of networking nodes;
2. limited qubit lifetimes, which imposes strict deadlines on how fast the data must be processed before it becomes useless;
3. noisy operations, which implies that applying operations on qubits degrades the quality of the qubit states themselves.

Nevertheless, quantum networking comes with an additional set of fundamental challenges related to time and data synchronization at various levels. At the physical layer, entanglement generation required both nodes involved to execute an operation at the same moment in time, with sub-nanosecond precision. At higher layers in the stack, this translates to the need for a network schedule that is shared among neighboring nodes — even though at these layers the granularity of the synchronized schedule can be coarser if the timing of the actual entanglement generation attempts can be fully coordinated at the physical layer. Moreover, the cross-node data dependencies inherent to many networking applications render time synchronization much more challenging, as the run-time schedule of a node may end up being delayed when said node needs to wait for information necessary to make logical decisions about next steps and such information comes from a remote party. Quantum networking applications that have cross-node data dependencies may incur more or less disruptive delays. Such applications are grouped into three categories of varying data synchronization implications by Van Meter et al. [40] — some applications allow post-selection of successful runs, some allow post-operation local corrections, and some require completion of certain operations before proceeding. These fundamental challenges can be summarized as:

4. cross-node scheduling dependencies, meaning that the operations on one node cannot be scheduled independently of other nodes;
5. data dependencies between various parts of a program, which require keeping quantum data alive in memory while waiting for an event to occur (in most cases, a message from a remote network node).

The fourth challenge is inherent to the nature of entanglement generation and to the physics of the devices currently in use. At this stage of development, the same quantum device functions as the processing unit and as the network device, and consequently local quantum operations (such as measurements and gates) and network operations cannot be performed simultaneously [42]. This limitation, however, could be mitigated by a new quantum hardware architecture separating the devices [43]. Finally, the fifth and last challenge is a fundamental issue, as it applies to quantum network applications regardless of technological progress. Challenges four and five are both arising from the need for information from a remote node at a certain level and with a certain degree of precision. However, the fourth challenge describes a distributed scheduling problem, whereas the fifth encapsulates two local requirements — OS-level locking mechanisms and "good" quantum memories. It is the last two challenges that fundamentally differentiate a quantum networking node from a quantum computing system, and are the key driver for a *networked* quantum node operating system.

2.3 Related Work

Relative to quantum networking, a substantial amount of software and systems work happens in the field of quantum computing. For example, operating systems for quantum computers (without networking functionality) are under active development in research and industry [24, 47]. Furthermore, extensive work exists on developing quantum computing architectures [8, 17, 30]. We aim to address new problems that arise specifically from the inclusion of quantum networking, which has not been considered at all in the aforementioned OSes and publications.

Nevertheless, systems research in quantum networking has been growing as a field as well. In particular, over the past several years there have been multiple proposals for quantum network protocol stacks [13, 21, 33, 41] and quantum network architectures [2, 15, 19, 25, 27, 28, 36]. One of the proposed network stacks has even been demonstrated experimentally on a state-of-the-art two-node network in a lab — as detailed in Chapter 4 — while the rest has only been validated in simulation. However, these works heavily focus on the network protocol aspects and whilst some of them acknowledge that the stacks will exist as a component in a bigger system, they do not tackle any of the related issues, such as resource management or task scheduling.

Quantum network applications themselves have also been demonstrated on small networks in laboratories [3]. However, such demonstrations have always been *ad hoc*, and scripted through low-level experimental controls as their purpose was to demonstrate hardware technology milestones rather than develop general systems for multiple users.

References

[1] M. H. Abobeih, J. Cramer, M. A. Bakker, N. Kalb, M. Markham, D. J. Twitchen, and T. H. Taminiau. "One-second Coherence for a Single Electron Spin Coupled to a Multi-qubit Nuclear-spin Environment". In: *Nature Commun.* 9.1 (2018), pp. 1–8. DOI: 10.1038/s41467-018-04916-z.

[2] A. Aguado, V. López, J. P. Brito, A. Pastor, D. R. López, and V. Martin. "Enabling Quantum Key Distribution Networks via Software-Defined Networking". In: *ONDM*. IEEE, 2020, pp. 1–5. DOI: 10.23919/ONDM48393.2020.9133024.

[3] S. Barz, E. Kashefi, A. Broadbent, J. F. Fitzsimons, A. Zeilinger, and P. Walther. "Demonstration of Blind Quantum Computing". In: *Science* 335.6066 (2012), pp. 303–308. DOI: 10.1126/science.1214707.

[4] M. Ben-Or and A. Hassidim. "Fast Quantum Byzantine Agreement". In: *STOC*. ACM, 2005, pp. 481–485. DOI: 10.1145/1060590.1060662.

[5] C. H. Bennett and G. Brassard. "Quantum Cryptography: Public Key distribution and Coin Tossing". In: *Theor. Comput. Sci.* 560.1 (2014), pp. 7–11. DOI: 10.1016/j.tcs.2014.05.025.

[6] 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". In: *Phys. Rev. Lett.* 70.13 (1993), pp. 1895–1899. DOI: 10.1103/PhysRevLett.70.1895.

[7] 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". In: *Nature* 497 (2013), pp. 86–90. DOI: 10.1038/nature12016.

[8] J. E. Bourassa, R. N. Alexander, M. Vasmer, A. Patil, I. Tzitrin, T. Matsuura, D. Su, B. Q. Baragiola, S. Guha, G. Dauphinais, K. K. Sabapathy, N. C. Menicucci, and I. Dhand. "Blueprint for a Scalable Photonic Fault-Tolerant Quantum Computer". In: *Quantum* 5 (2021), pp. 392–430. DOI: 10.22331/q-2021-02-04-392.

[9] M. Bozzio, U. Chabaud, I. Kerenidis, and E. Diamanti. "Quantum Weak Coin Flipping With a Single Photon". In: *Phys. Rev. A* 102 (2020), p. 022414. DOI: 10.1103/PhysRevA.102.022414.

[10] 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". In: *Phys. Rev. X* 9.3 (2019), pp. 031045-1–031045-12. DOI: 10.1103/PhysRevX.9.031045.

[11] A. Broadbent, J. Fitzsimons, and E. Kashefi. "Universal Blind Quantum Computation". In: *FOCS*. IEEE, 2009, pp. 517–526. DOI: 10.1109/FOCS.2009.36.

[12] A. M. Childs. "Secure Assisted Quantum Computation". In: *Quantum Inf. Comput.* 5.6 (2005), pp. 456–466. DOI: 10.26421/QIC5.6-4.

[13] A. Dahlberg, M. Skrzypczyk, T. Coopmans, L. Wubben, F. Rozpędek, M. Pompili, A. Stolk, P. Pawełczak, R. Knegjens, J. de Oliveira Filho, R. Hanson, and S. Wehner. "A Link Layer Protocol for Quantum Networks". In: *SIGCOMM*. ACM, 2019, pp. 159–173. DOI: 10.1145/3341302.3342070.

[15] S. DiAdamo, B. Qi, G. Miller, R. Kompella, and A. Shabani. "Packet Switching in Quantum Networks: A Path to the Quantum Internet". In: *Phys. Rev. Res.* 4.4 (2022), p. 043064. DOI: 10.1103/PhysRevResearch.4.043064.

[16] A. K. Ekert. "Quantum Cryptography Based on Bell's Theorem". In: *Phys. Rev. Lett.* 67.6 (1991), pp. 661–663. DOI: 10.1103/PhysRevLett.67.661.

[17] X. Fu, M. A. Rol, C. C. Bultink, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudever, L. DiCarlo, and K. Bertels. "An Experimental Microarchitecture for a Superconducting Quantum Processor". In: *MICRO*. ACM, 2017, pp. 813–825. DOI: 10.1145/3123939.3123952.

[18] D. Gottesman, T. Jennewein, and S. Croke. "Longer-Baseline Telescopes Using Quantum Repeaters". In: *Phys. Rev. Lett.* 109.7 (2012), pp. 070503-1–070503-5. DOI: 10.1103/PhysRevLett.109.070503.

[19] H. Gu, Z. Li, R. Yu, X. Wang, F. Zhou, and J. Liu. "FENDI: High-Fidelity Entanglement Distribution in the Quantum Internet". 2023. arXiv: 2301.08269.

[20] 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". In: *Nature* 558.7709 (2018), pp. 268–273. DOI: 10.1038/s41586-018-0200-5.

[21] J. Illiano, M. Caleffi, A. Manzalini, and A. S. Cacciapuoti. "Quantum Internet Protocol Stack: a Comprehensive Survey". 2022. arXiv: 2202.10894.

[22] H. J. Kimble. "The Quantum Internet". In: *Nature* 453.7198 (2008), pp. 1023–1030. DOI: 10.1038/nature07127.

[23] P. Kómár, E. M. Kessler, M. Bishof, L. Jiang, A. S. Sørensen, J. Ye, and M. D. Lukin. "A Quantum Network of Clocks". In: *Nature Phys.* 10.8 (2014), pp. 582–587. DOI: 10.1038/nphys3000.

[24] W. Kong, J. Wang, Y. Han, Y. Wu, Y. Zhang, M. Dou, Y. Fang, and G. Guo. "Origin Pilot: a Quantum Operating System for Effecient Usage of Quantum Resources". 2021. arXiv: 2105.10730. URL: https://arxiv.org/abs/2105.10730.

[25] Z. Li, K. Xue, J. Li, N. Yu, D. S. L. Wei, and R. Li. "Connection-Oriented and Connectionless Remote Entanglement Distribution Strategies in Quantum Networks". In: *IEEE Network* 36.6 (2022), pp. 150–156. DOI: 10.1109/MNET.107.2100483.

[26] W.-Z. Liu, Y.-Z. Zhang, Y.-Z. Zhen, M.-H. Li, Y. Liu, J. Fan, F. Xu, Q. Zhang, and J.-W. Pan. "Toward a Photonic Demonstration of Device-Independent Quantum Key Distribution". In: *Phys. Rev. Lett.* 129.5 (2022), p. 050502. DOI: 10.1103/PhysRevLett.129.050502.

[27] R. Mandil, S. DiAdamo, B. Qi, and A. Shabani. "Quantum Key Distribution in a Packet-Switched Network". 2023. arXiv: 2302.14005.

[28] T. Matsuo, C. Durand, and R. Van Meter. “Quantum Link Bootstrapping Using a RuleSet-Based Communication Protocol”. In: *Phys. Rev. A* 100.5 (2019), p. 052320. DOI: 10.1103/PhysRevA.100.052320.

[29] D. L. Moehring, P. Maunz, S. Olmschenk, K. C. Younge, D. N. Matsukevich, L.-M. Duan, and C. Monroe. “Entanglement of Single-Atom Quantum Bits at a Distance”. In: *Nature* 449 (2007), pp. 68–71. DOI: 10.1038/nature06118.

[30] P. Murali, N. M. Linke, M. Martonosi, A. J. Abhari, N. H. Nguyen, and C. H. Alderete. “Full-Stack, Real-System Quantum Computer Studies: Architectural Comparisons and Design Insights”. In: *ISCA*. ACM, 2019, pp. 527–540. DOI: 10.1145/3307650.3322273.

[31] N. H. Y. Ng, S. K. Joshi, C. Chen Ming, C. Kurtsiefer, and S. Wehner. “Experimental Implementation of Bit Commitment in the Noisy-Storage Model”. In: *Nature communications* 3.1 (2012), pp. 1–7. DOI: 10.1038/ncomms2268.

[32] M. A. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. American Association of Physics Teachers, 2002.

[33] A. Pirker and W. Dür. “A Quantum Network Stack and Protocols for Reliable Entanglement-Based Networks”. In: *New Journal of Physics* 21.3 (2019), p. 033003. URL: 10.1088/1367-2630/ab05f7.

[34] 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”. In: *Science* 372.6539 (2021), pp. 259–264. DOI: 10.1126/science.abg1919.

[35] M. Pompili, C. Delle Donne, I. te Raa, B. van der Vecht, M. Skrzypczyk, G. M. 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”. In: *npj Quantum Information* 8.1 (2022), p. 121. DOI: 10.1038/s41534-022-00631-2.

[36] S. Pouryousef, N. K. Panigrahy, and D. Towsley. “A Quantum Overlay Network for Efficient Entanglement Distribution”. 2022. arXiv: 2212.01694.

[37] A. Reiserer and G. Rempe. “Cavity-based Quantum Networks with Single Atoms and Optical Photons”. In: *Rev. Mod. Phys.* 87.4 (2015), pp. 1379–1418. DOI: 10.1103/RevModPhys.87.1379.

[38] C. Thalacker, F. Hahn, J. de Jong, A. Pappa, and S. Barz. “Anonymous and Secret Communication in Quantum Networks”. In: *New Journal of Physics* 23.8 (2021). DOI: 10.1088/1367-2630/ac1808.

[39] G. Vallone, D. Bacco, D. Dequal, S. Gaiarin, V. Luceri, G. Bianco, and P. Villoresi. “Experimental Satellite Quantum Communications”. In: *Phys. Rev. Lett.* 115.4 (2015), pp. 040502-1–040502-5. DOI: 10.1103/PhysRevLett.115.040502.

[40] R. Van Meter, T. Satoh, S. Nagayama, T. Matsuo, and S. Suzuki. “Optimizing Timing of High-Success-Probability Quantum Repeaters”. 2017. arXiv: 1701.04586.

[41] R. Van Meter and J. Touch. “Designing Quantum Repeater Networks”. In: *IEEE Communications Magazine* 51.8 (2013), pp. 64–71. DOI: 10.1109/MCOM.2013.6576340.

[42] G. Vardoyan, S. Guha, P. Nain, and D. Towsley. “On the Stochastic Analysis of a Quantum Entanglement Switch”. In: *Perform. Eval. Rev.* 47.2 (2019), pp. 27–29. DOI: 10.1145/3374888.3374899.

[43] G. Vardoyan, M. Skrzypczyk, and S. Wehner. “On the Quantum Performance Evaluation of two Distributed Quantum Architectures”. In: *Performance Evaluation* 153 (2022), pp. 1–26. DOI: 10.1016/j.peva.2021.102242.

[44] S. Wehner, D. Elkouss, and R. Hanson. “Quantum Internet: A Vision for the Road Ahead”. In: *Science* 362.6412 (2018), pp. 1–9. DOI: 10.1126/science.aam9288.

[45] J. Yin, Y. Cao, Y.-H. Li, S.-K. Liao, L. Zhang, J.-G. Ren, W.-Q. Cai, W.-Y. Liu, B. Li, H. Dai, G.-B. Li, Q.-M. Lu, Y.-H. Gong, Y. Xu, S.-L. Li, F.-Z. Li, Y.-Y. Yin, Z.-Q. Jiang, M. Li, J.-J. Jia, G. Ren, D. He, Y.-L. Zhou, X.-X. Zhang, N. Wang, X. Chang, Z.-C. Zhu, N.-L. Liu, Y.-A. Chen, C.-Y. Lu, R. Shu, C.-Z. Peng, J.-Y. Wang, and J.-W. Pan. “Satellite-Based Entanglement Distribution Over 1200 Kilometers”. In: *Science* 356.6343 (2017), pp. 1140–1144. DOI: 10.1126/science.aan3211.

[46] W. Zhang, T. van Leent, K. Redeker, R. Garthoff, R. Schwonnek, F. Fertig, S. Eppelt, W. Rosenfeld, V. Scarani, C. C.-W. Lim, et al. “A Device-Independent Quantum Key Distribution System for Distant Users”. In: *Nature* 607.7920 (2022), pp. 687–691. DOI: 10.1038/s41586-022-04891-y.

[47] *Deltaflow.OS*. Riverlane. URL: https://www.riverlane.com/products/ (visited on Feb. 28, 2023).

3 Architecture of an Operating System for a Quantum Network Node

The end goal of an operating system (OS) for quantum network nodes is to bridge the gap between user applications — written in high-level and platform-independent software — and the underlying quantum hardware, to which the user is agnostic. How can one design a control system that adheres to this objective, while addressing the challenges that come with quantum networking? And what does an example architecture of such a system look like? This chapter explores the cardinal design considerations that should drive the design of an OS for quantum network nodes, and proposes a proof-of-principle architecture for such an OS.

This chapter is extracted from the article in preparation: **C. Delle Donne**, M. Iuliano, B. van der Vecht, M. Skrzypczyk, I. te Raa, G. M. Ferreira, T. van der Steenhoven, A. R.-P. Montblanch, M. Pompili, S. L. N. Hermans, N. Demetriou, B. van Ommen, T. H. Taminiau, P. Pawełczak, W. Kozlowski, R. Hanson, and S. Wehner. "QNodeOS: An Operating System for Quantum Network Nodes".

Contributions: as indicated in Chapter 2.

An operating system (OS) is the cornerstone of a system's control software: it marshals access to physical resources, abstracts low-level hardware functionalities into user-friendly services, and provides an interface to users to program and run applications on the system. Our goal here is to apply basic principles from classical OS design literature to our novel use case of programmable and scalable quantum networking nodes, and to (hopefully) create a framework where the challenges outlined in Chapter 2 can be studied and addressed. We thus investigate the general requirements that such a system should satisfy, illustrated with the example of a quantum processor based on nitrogen-vacancy (NV) centers in diamond. This provides a guideline for future systems of this form. We also propose the *first proof-of-principle architecture for an operating system for quantum network nodes*, which we call QNodeOS. Our system's capabilities include quantum memory management, scheduling different types of quantum operations on the device, as well as an interface to different drivers addressing several possible quantum hardware architectures. On the quantum networking front, QNodeOS adopts the quantum network stack and protocols proposed by Dahlberg et al. [2] and by Kozlowski et al. [8].

3.1 General Design Considerations

We assume that the OS builds upon a quantum hardware system capable of the execution of *physical instructions* addressing specific qubits on the quantum chip. These physical instructions may be dependent on the type of quantum hardware (e.g. NV in diamond, or ion traps), and include instructions for initializing and measuring qubits on the chip, moving the state of a qubit to another location in the quantum memory, performing quantum gates, as well as to make attempts at entanglement generation at the physical layer [10]. The quantum hardware furthermore exposes the capabilities of the quantum chip: (1) the number of qubits (2) the type of each qubit (3) the memory lifetime of the qubits (4) the physical instructions that can be performed on on the qubit(s) and (5) the average quality of these instructions (Chapter 2). We emphasize that, unlike in classical computing, there is currently no established low-level microarchitecture that defines the line between (quantum) hardware and software upon which such an OS would be built. We nevertheless expect that almost all of the below would be functions taken on by any OS, some of which could possibly be shifted to control hardware in the future.

Each node in the network runs its own independent quantum network OS. Nodes may interact with each other using both classical message passing as well as entanglement generation. The goal of the combined system is to execute quantum network applications, which themselves consist of separate programs running on the OS of two (or more) network nodes. Such programs generally also communicate via classical message passing and entanglement generation. Each program itself consists of both classical and quantum blocks of code, where the quantum blocks of code may contain low-level classical logic (specifically, branching on classical variables and loops). Classical blocks of code may depend on quantum ones via classical variables generated during the quantum execution (measurement results, notification of entanglement generation, and information on the state of the quantum system such as the availability of qubits). Similarly, quantum blocks may depend on variables set by the classical blocks, such as messages received from remote network nodes. Finally, quantum blocks may themselves depend on other quantum blocks via qubits in the quantum memory.

Whilst the OS should understand dependencies and provide mechanisms to specify them, a non-goal of the OS is to identify what is a classical block and what is a quantum block — decomposing a program is not necessarily a trivial task, and might result in unwarranted latency at the OS level. Similarly, we assume that it is not the OS's responsibility to determine deadlines or priorities in the execution, given that setting deadlines (e.g. when too much time has elapsed for the qubits to be useful) is in general a computationally expensive procedure, sometimes estimated by a repeated simulation of the execution. Instead, program decomposition and deadline estimation would be the compiler's job, which can leverage application-level information and knowledge of the quantum hardware (e.g. memory lifetimes) to carry out these tasks. At the compiler level, latency is much less an issue, and thus computing block boundaries and deadlines can be performed without stringent time constraints. We remark that — especially in the near term — some of these tasks might require the programmer's intervention, at least until compilers for quantum networking applications become more sophisticated. Specifying the compiler's and the programmer's exact responsibilities is outside the scope of this work.

3.2 Key OS Components

We now describe the essential components we envision any OS for quantum network nodes to have.

3.2.1 Memory Management Unit

Executing quantum network applications demands a continuing interaction between the classical and quantum parts of the execution, including keeping qubits alive in memory to take further actions depending on messages from remote network nodes. We thus require persistent memory management capabilities. This may be taken up by a *quantum memory management unit* (QMMU). A QMMU has knowledge of the physical qubits available on the underlying quantum hardware, and may store metadata such as qubit type (communication or storage qubit) and qubit lifetime. A QMMU allows physical qubits to be assigned to different applications or to the OS itself, and may allow a transfer of ownership of the qubits from one owner to another. A QMMU may also provide abstractions familiar to classical computing such as a virtual address space, where the applications refer to virtual qubit addresses that are then translated to physical qubit addresses. This avoids the situation in which physical qubit addresses must be bound at compile time, particularly limiting when allowing multiple applications to concurrently run on the same node. Advanced forms of a QMMU may also cater to the limitations of near term quantum devices, by matching memory lifetime requirements specified by the application code to the capabilities of the underlying qubits, as well their topology. While one cannot measure the decoherence of a qubit during a general program execution on the quantum level, the QMMU could also take into account additional information from the classical control system to signal to the application that a qubit has become invalid.

3.2.2 Quantum Network Stack

The OS should include the capability for quantum communication with remote nodes in the network, typically the generation of entanglement. We thus assume that the OS realizes a quantum network stack that can be relied upon to enable entanglement generation, where we refer to Ref. [2] for design considerations of quantum network stacks themselves. The network stack allows an application to request a certain number or rate of entangled pairs to be produced with remote nodes with a specified quality (i.e. fidelity) of entanglement. The stack is responsible for ensuring the delivery of the entanglement. One possible quantum network stack can be found in Ref. [2] including the first link layer protocol now realized on quantum hardware [10] (as described in Chapter 4), and a network layer protocol (as proposed in Ref. [8]).

To successfully produce entanglement, the network stack needs access to a communication qubit, resulting in two requirements for the rest of the system:

1. A scheduler should take into account that generating entanglement at the physical layer between two nodes directly connected by a physical communication medium requires that the two nodes apply a series of physical operations with very precise timing synchronization between them (nanosecond precision with sub-nanosecond jitter). Therefore, entanglement generation across a link with an adjacent node must always be scheduled in a synchronized manner between the two adjacent neighbors. Similarly, due to limited memory lifetimes, generating entanglement with the help of an intermediary node at the network layer [8] requires specific operations (entanglement swapping) to be scheduled at all three nodes within a time window allowed by the memory lifetimes.
2. On some quantum hardware systems (e.g. NV in diamond), the communication qubit is in general needed to enable the execution of quantum gates on and in-between storage qubits. This has implications both on the scheduler (local instructions cannot be scheduled concurrently to networked ones), as well as on the QMMU, which needs to allow qubit ownership transfer between applications and the network stack. A typical use case of such ownership transfer would occur when the network stack claims the communication qubit for entanglement generation, and then yields it to an application.

3.2.3 Scheduler

In order to maximize the usage of resources, we envision the OS to include a scheduler. This may be a single scheduler, or more likely several schedulers that address scheduling at different levels. In general, we may consider scheduling at the level of applications, at the level of blocks of quantum code, and at the level of instructions, each level not being independent of one another.

General considerations. A scheduler for quantum network nodes should be capable of managing the limited physical resources to achieve the desired performance. The performance of any form of scheduling method in the quantum domain is assessed not only by existing classical metrics — like throughput and latency — but also by quantum metrics (see Chapter 2). At the level of the application, latency can be measured in terms of the success probability of the quantum network application. At the level of an operation (or a block of operations) it may be measured by the quality (fidelity) of the quantum states and operations performed. We remind that due to limited memory lifetimes, delays have always a direct impact on the quantum performance, resulting in general in trade-offs between classical and quantum performance metrics when assessing any scheduler.

Practically, the scheduler in question should allocate the underlying physical resources — most importantly, the qubits — based on a set of well-defined constraints:

1. *Synchronized network schedule*: due to the bilateral nature of entanglement, each node will have its quantum networking activity synchronized with its neighbors, meaning that a missed synchronization window on one node results in a waste of resources on remote nodes too.
2. *Local quantum computation*: in addition to quantum networking, a node's resources must also be reserved for local quantum gates, which are integral parts of quantum networking applications.
3. *Inter-block dependencies*: quantum and classical processing blocks of an application may depend on results originating from other blocks, and thus cannot be scheduled independently.
4. *Multitasking*: for a node to be shared by multiple users, the scheduler should not allocate all the available resources to a single application indefinitely, and instead it should be aware of the presence of multiple applications and multiple users.

Additionally, scheduling at any level could optionally process another set of input variables, where we generally assume that the compiler provides aggregate advice based on these input variables to the OS:

1. *Duration of operations*: local quantum operations typically take a fixed amount of time and always succeed. Entanglement, on the other hand, is a probabilistic process, and generating an entangled pair can take an indefinite (and large) number of attempts. Scheduling decisions may factor this in to yield better performance.
2. *Decoherence*: as already stated, the fidelity of a quantum state stored in a qubit is not constant, and it also degrades due to physical noise induced by other qubits and by operations applied on such qubits. An advanced scheduler could use knowledge of qubit lifetimes and elapsed time to dynamically re-prioritize application demands based on the advice of the compiler.

Scheduling of applications. In an OS allowing the execution of concurrent quantum network applications, the task of an application-level scheduler would be to decide which application to schedule next. We remark that a compiler aware of the underlying capabilities of the hardware system (e.g. memory lifetimes) can provide advice in the form of a deadline by which the network application must have completed in order to be successful. To allow for potentially time-consuming classical pre- and post-processing, it is natural to apply such deadlines not for the entirety of the application, but for the period between initializing the qubits and terminating the quantum part of the execution. This suggests in general using real-time schedulers for quantum network applications, taking inspiration from the extensive work on this topic in classical systems (see e.g. Ref. [9]). While outside the scope of this work, we remark that this type of scheduling offers to inspire interesting new work in a form of "quantum soft-real time" scheduling, where deadlines may occasionally be missed at the expense of reduced application performance (success probability), to maximize the overall performance of the system in which applications are typically executed repeatedly. A benchmark for the quantum performance of any application-level scheduler is the quality of the quantum execution when the entire system (all nodes) are reserved for only one application at the time.

Scheduling of quantum blocks. Scheduling can also (additionally) be performed on the level of quantum blocks of code. This can in principle also take the form of a (soft) real-time scheduler that schedules blocks of the currently running application, or schedules blocks of several applications (potentially independently of any application level scheduling) depending on the availability of resources on the quantum hardware system. This form of scheduling may be appealing for efficiency reasons, depending on where what parts of the OS are executed, where some parts are closer to the underlying hardware system than others (see e.g. Section 3.3).

Scheduling of operations. Finally, scheduling can be performed at two levels of operations: First, one can consider the problem of scheduling local versus networked instructions, where one simple way of realizing a schedule that respects the constraints inherent to such a schedule is presented in Section 3.3. Second, one can consider scheduling any form of operation on the underlying quantum processor. While our current realization of QNodeOS achieves this by populating an instruction queue in software, we envision that this form of scheduling would later be moved from QNodeOS to a hardware module in a microarchitecture for quantum network nodes, as for instance in the work by Fu et al. [6].

3.3 QNodeOS Design

QNodeOS is an OS for quantum network nodes, designed to address the challenges described in Chapter 2. It includes all the identified key components, plus some additional convenience abstraction layers. The current design of QNodeOS is considered *best-effort* — it is meant to explore the main design aspects of an OS for quantum networks, and to provide a minimum working system.

3.3.1 Full Stack of a Quantum Network Node

As described in Chapter 2 and illustrated in Figure 2.2, a quantum network application consists of programs running on different end nodes, composed of blocks of quantum code and blocks of fully-classical code. In fact, quantum code blocks may also contain simple classical logic — like simple arithmetic and branching instructions — used for flow control. These blocks do not have any dependencies on data originating from other nodes. Fully-classical code blocks — which include local processing and communication with other end nodes — mainly produce input data for the next quantum code blocks. That is, a classical code block typically precedes a quantum code block whose instructions depend on external data coming from a remote end node. In our implementation, quantum code blocks are expressed in *NetQASM* [3]. NetQASM is an open-source software development kit (SDK) and instruction set for quantum applications [12]. The NetQASM SDK compiles a quantum network application, written in Python, into a series of classical and quantum code blocks. The instruction set used for the quantum code blocks is similar to other QASM languages [1, 5, 7], but it is extended to include instructions for quantum networking. NetQASM is not a strict requirement of QNodeOS — one can implement it to support other low-level languages as well — but it does impose certain conventions (described in Ref. [3]) on a particular implementation of the system.

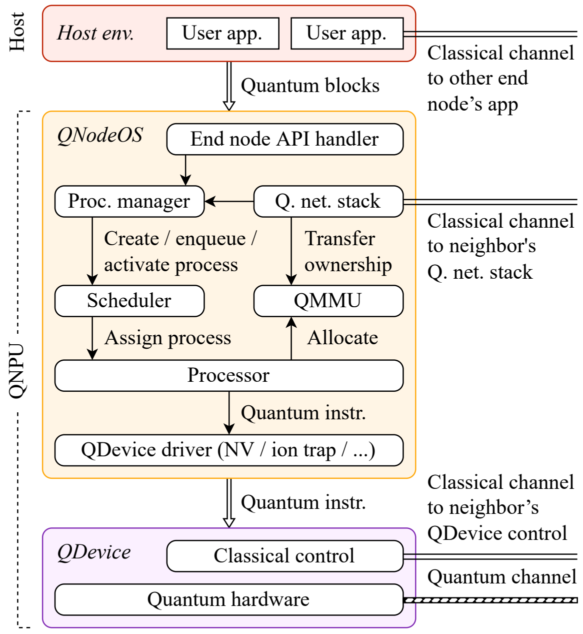


Figure 3.1: Full-stack architecture of a quantum network node. User applications start in the host environment, which runs classical code blocks and offloads quantum code blocks to the QNPU. QNodeOS, lying at the top of the QNPU, processes quantum code blocks and invokes the quantum device (QDevice) to run the actual quantum instructions. QNodeOS consists of an end-node API handler, a quantum network stack (Q. net. stack), a process manager (Proc. manager), a quantum memory management unit (QMMU), a scheduler, a processor, and a QDevice driver to communicate with the QDevice itself. The host shares a classical communication channel with other end nodes' hosts for application data. QNodeOS shares a classical channel with its neighbors. The QDevice shares a classical channel for coordination and a quantum channel for entanglement with its neighbors.

In principle, classical and quantum code blocks can be run on a single system, provided that this has a connection to the quantum device to execute the actual quantum instructions. However, in the interest of a simpler implementation, where each system has a scoped responsibility, we opted to map classical and quantum blocks onto two distinct environments. Classical blocks are run on a system that features a fully-fledged OS (like Linux), with access to high level programming languages (like C++ and Python) and libraries. Quantum blocks are delegated to the *quantum network processing unit* (QNPU), which is a system capable of interpreting quantum code blocks and managing the resources of a quantum device. In our design, a quantum network application starts on the general-purpose OS — that we call the *host* — which runs classical code blocks internally, and offloads quantum code blocks to the QNPU. QNodeOS is the topmost component of the QNPU. It runs the quantum code blocks, relying on the underlying quantum device — denoted as *QDevice* — to execute the actual quantum operations.

The architecture of a quantum network node is depicted in Figure 3.1. An alternative architecture could merge host and QNodeOS into the same system, potentially enabling some performance optimizations, at the cost of a higher system complexity. We also note that the host, QNodeOS and the classical control modules of the QDevice can be deployed on distinct physical devices, or combined in some way.

3.3.2 Processes

A quantum network application starts on the host — there, the host environment compiles it into classical and quantum code blocks, and creates a new process associated with the application. The host then registers the application with QNodeOS (through QNodeOS's end-node API), which, in turn, creates its own process associated with the registered application. The process on the host is a standard OS process, which executes the classical code blocks and interacts with the counterpart process on QNodeOS (by means of a shared memory, as defined in NetQASM [3]). On QNodeOS, a process encapsulates the execution of quantum code blocks of an application with associated context information, such as process owner, ID, process state and priority.

The execution time of an application is typically dominated by that of quantum blocks, as entanglement generation is a time-consuming operation, and its duration grows exponentially with the distance between the nodes. For this reason, in this work we focus on the scheduling of quantum blocks only, and thus we only discuss QNodeOS processes from this point onward. Again, this does not exclude that, in a future iteration of the design, host and QNodeOS could be merged into one system, and therefore classical and quantum blocks would be scheduled jointly.

User processes. QNodeOS allocates a new *user process* to each quantum network application registered by the host. A user process becomes active (ready to be scheduled) as soon as QNodeOS receives a quantum code block from the host. Multiple user processes — relative to different host applications — can be concurrently active on QNodeOS, but only one can be running at any time (multi-core support is in principle possible, but not included in this version of QNodeOS). A running user process executes its quantum code block directly, except for entanglement requests, which are instead submitted to the quantum network stack and executed asynchronously.

Network process. QNodeOS also defines *kernel processes*, which are similar to user processes, but are created by default (on boot) and have different priority values. Currently, the only existing kernel process is the *network process*. The network process, owned by the quantum network stack, handles entanglement requests submitted by user processes, coordinates entanglement generation with the rest of the network, and eventually returns entangled qubits to user processes. The activation of the network process is dictated by a network-wide entanglement generation schedule. Such a schedule defines when a particular entanglement generation request can be processed, and therefore it has intersecting entries on adjacent nodes (given that entanglement is a two-party process). The schedule can be computed by a centralized network controller [11] or by a distributed protocol [2]. In our design, the network process follows a *time-division multiple access schedule*, computed by a centralized network controller (as originally proposed by Skrzypczyk and Wehner [11]) and installed on each QNodeOS node.

Process states. A QNodeOS process can be in any of the following states: (1) *idle*: when it exists but it is not active; (2) *ready*: when it is active and ready to issue instructions; (3) *running*: when it is running on QNodeOS; (4) *waiting*: when it is waiting for some event to occur. User processes enter the waiting state when they need one or more entangled

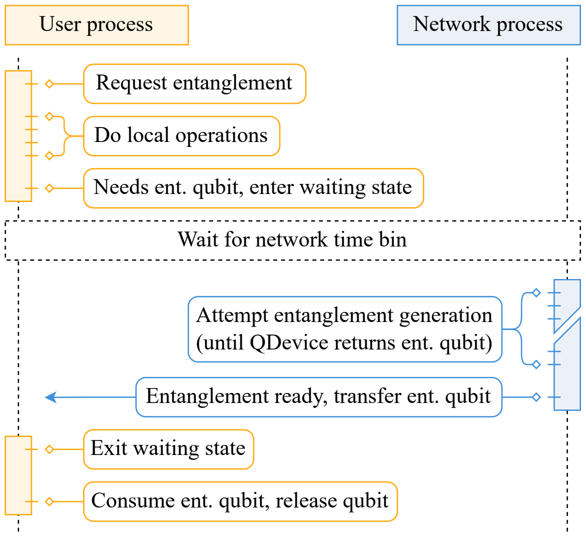

User process
Network process
Request entanglement
Do local operations
Needs ent. qubit, enter waiting state
Wait for network time bin
Attempt entanglement generation (until QDevice returns ent. qubit)
Entanglement ready, transfer ent. qubit
Exit waiting state
Consume ent. qubit, release qubit

Figure 3.2: Flow of execution between a user process requesting entanglement and the network process responsible for generating entanglement. The user process starts by issuing an asynchronous entanglement request. Once issued, it is free to continue with other local operations or classical processing. Once it reaches a point in its execution where entanglement is required the process enters the waiting state. The network process is then scheduled once the appropriate time bin starts, as determined by the network schedule. Once it is running, the network process attempts entanglement generation until entanglement success (or until a set timeout). The entangled qubit is then transferred to the user process. This unblocks the process which consumes the entanglement and releases the qubit.

pairs to proceed, and become ready again once all the requested pairs are delivered by the network process.

Inter-process communication. At the moment, QNodeOS does not allow for any explicit inter-process communication. The only indirect primitive available to processes to interact with one another is *qubit ownership transfer*, used when a process produces a qubit state which is to be consumed by another process. In particular, the network process transfers ownership of the entangled qubits that it produces to the process which requested them.

Process concurrency. The strict separation between local quantum processing and quantum networking is a key design decision in QNodeOS, as it helps us address the scheduling challenge described in Chapter 2. A user process can continue executing local instructions even after it has requested entanglement. Conversely, networking instructions can execute asynchronously of local quantum instructions. This is rather important in a quantum network, since entanglement generation must be synchronized with the neighboring node (and possibly the rest of the network [11]). Additionally, separating user applications into user processes also allows QNodeOS to schedule several applications *concurrently*.

Process flow. Figure 3.2 illustrates the typical control flow between a user process and the network process. User processes are free to execute any non-networked instructions independently of the network process and other user processes. Once the application

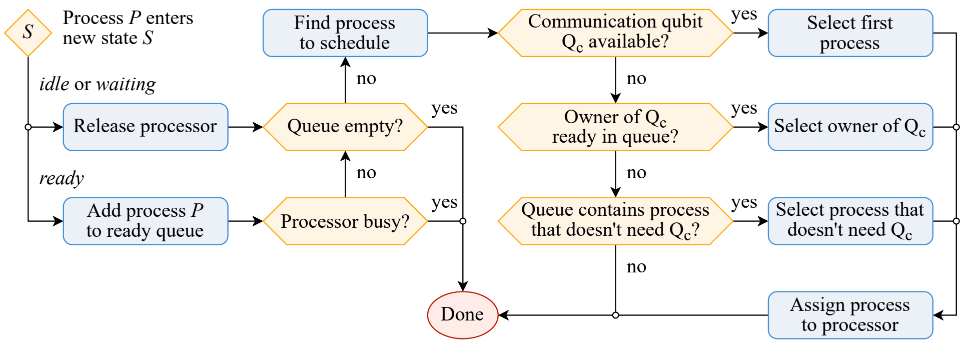

S
Process P enters new state S
idle or waiting
ready
Release processor
Add process P to ready queue
Queue empty?
Processor busy?
Find process to schedule
no
yes
Communication qubit Q_c available?
Owner of Q_c ready in queue?
Queue contains process that doesn't need Q_c?
Select first process
Select owner of Q_c
Select process that doesn't need Q_c
Assign process to processor
Done

Figure 3.3: Flowchart representation of the scheduling algorithm employed by QNodeOS. The scheduler is activated when a process P transitions to a new state S. If S is *idle* or *waiting*, the process has (temporarily) terminated its execution, and thus relinquished the processor, in which case the scheduler can proceed. If S is *ready*, the process is added to the prioritized ready queue, and if the processor is not already busy, the scheduler can proceed again. When the ready queue is not empty, the scheduler looks for the next scheduling candidate as follows: (1) if the communication qubit is available, the scheduler simply selects the highest-priority process in the ready queue; (2) otherwise, if the process owning the communication qubit is waiting in the ready queue, the scheduler selects this process as the scheduling candidate; (3) as a last resort, the scheduler searches the ready queue for a process that does not need the communication qubit. At the moment, the scheduling algorithm is tuned to NV center-based quantum platforms, where the communication qubit plays a rather central role in both entanglement generation and local quantum computation.

reaches a point in its execution where an entangled qubit is required, the process enters the waiting state and is flagged as waiting for entanglement. When the network process is scheduled, it issues network instructions and generates entanglement as requested by the user process. Once an entangled pair is generated by the network process, the qubit is handed over to the waiting user process. When all the entangled pairs that the user process was waiting for are delivered, the user process becomes ready and can start running again.

Process control block. Each process has a *process control block* (PCB) assigned to it. The PCB stores metadata including process ID, process priority, process state and process data. Additionally, the PCB of a process maintains a flag that is set when the process tries to issue a quantum instruction that needs the communication qubit and such a qubit is not available. This flag is checked by the scheduler when a process needs to be selected for execution, to avoid scheduling processes that need a resource that is not available.

3.3.3 Process Scheduling

As previously mentioned, we focus our attention on the quantum blocks of an application, and thus we only discuss the scheduling of QNodeOS processes. At present, the QNodeOS scheduler does not give any guarantees on when a process is scheduled — for that, one would need to define concrete real-time constraints to feed to the scheduler. Instead, the current version of QNodeOS implements a scheduler that offers a *best-effort* service, where processes are selected on the basis of their priority, and preemption is not allowed. The scheduling algorithm is represented in the flowchart in Figure 3.3.

Priority scheduling. QNodeOS schedules ready processes using a *priority-based* algorithm. In particular, the network process is assigned the highest priority, and is given precedence whenever the network schedule activates it [11]. Prioritizing entanglement generation over local operations is key for a node to be able to fulfill its networking duty, and to avoid peer nodes to waste their resources.

Ready queue. The main data structure supporting the functioning of the scheduler is a *ready queue*. This is essentially an ordered list of processes ready to be selected for execution. When a process becomes ready, it is inserted into the appropriate location in the queue, according to the properties specified in its PCB. The ready queue is sorted by process priority. Processes of the same priority are stored first-in first-out.

Shared resources. The most important resources to be shared among processes are qubits — in particular, the *communication qubit*. In the current design, there is only one ready queue. The scheduler selects the queue's head for execution, unless that process needs the communication qubit and this is not available — in which case, the queue is searched either for the communication qubit's owner process, or for a process that does not need the communication qubit. An alternative design could have two ready queues, where one is dedicated to processes that do not need the communication qubit. In this case, when the communication qubit is available, the scheduler would need to look up both queues, and select the appropriate process according to priority and time of arrival. In either case, the scarce availability of the shared resource (the communication qubit) might result in the high-priority network process to become blocked until a lower-priority user process relinquishes the resource, if such a process owns the resource itself. In a more sophisticated design, one should make sure that low-priority processes always relinquish the shared resource within a finite amount of time.

No preemption. To avoid context switching overhead, potentially leading to degraded fidelity, the QNodeOS scheduler is *cooperative*. That is, once a process is scheduled, it gets to run until it either completes all of its instructions or it blocks waiting for entanglement. Processes of the same priority are selected first-in first-out, and they are not executed in a round-robin manner — again, that would require preemption. Allowing process preemption would need a definition of critical section and could potentially impact the quality of the affected qubit states. However, no preemption means that the execution of the high-priority network process might get delayed by a long-running user process. Again, a future version of QNodeOS could time-cap the execution of user processes when these are blocking the network process.

3.3.4 QNodeOS Architecture

We refer again to Figure 3.1, which illustrates the internals of QNodeOS, and outlines the interactions with the rest of the components of a quantum network node. At its top, QNodeOS implements an *end-node API handler* to process requests from the host. Internally, QNodeOS features a *quantum network stack*, a *process manager*, a *process scheduler*, a *quantum memory management unit* (QMMU), and an *instruction processor*. Actual quantum instructions are offloaded to the underlying quantum device (QDevice) through the *QDevice driver*.

We note that QNodeOS itself is an entirely classical system that interacts with the quantum hardware (the QDevice). At the moment, our implementation of QNodeOS is fully software, including the instruction processor. In general, the system may be implemented entirely in software running on a classical CPU, or parts of its functionality may be implemented in classical hardware (e.g. FPGA or ASIC).

End-node API. Each user application is registered on QNodeOS by the host through the end-node API. Using the same API, the host can then send quantum code blocks and receive their results (like measurement outcomes and entanglement generation information). Upon registration of an application, QNodeOS allocates a new user process. Upon reception of a quantum code block, the related user process is activated and made eligible for scheduling.

Process manager, scheduler, processor. The QNodeOS process manager keeps track of existing user and kernel processes and their execution context. Upon activation, processes are added to a scheduling queue. When selected by the scheduler, a process is assigned to the QNodeOS processor, which (1) executes classical control-flow instructions directly, (2) offloads local quantum computation to the QDevice, and (3) registers entanglement requests with the quantum network stack.

Quantum network stack. The role of the quantum network stack in QNodeOS is to abstract the unreliable entanglement attempts that the QDevice offers into a robust, multi-node network service. The network stack can handle entanglement generation requests which specify a number of parameters — including source and destination, desired fidelity, and number of entangled pairs — and returns the entangled pair(s) described by an identifier and the generated Bell state. The quantum network stack owns the network process, whose activation is dictated by a network-wide entanglement schedule. The quantum network stack in QNodeOS is based on the model outlined by Dahlberg et al. [2], and features a *link layer protocol* — presented in the same work, and recently evaluated on hardware [10] (Chapter 4) — and a *network layer protocol* — as designed by Kozlowski et al. [8].

Quantum memory management unit. QNodeOS's QMMU implements basic memory management functionality: *virtual address spaces* and *qubit ownership transfer*. A virtual quantum memory address space is akin to a classical virtual address space, but it isolates the qubit address spaces of QNodeOS processes. Ownership transfer is an indirect type of inter-process communication (IPC) mechanism for passing quantum data between processes. Since quantum states cannot be copied due to the no-cloning theorem, this is the only valid IPC for passing quantum data between address spaces. Ownership transfer is only logical — only the data's owner is updated — rather than it being a physical move of quantum data in the memory. This way we can avoid issuing a QDevice instruction which would cause degradation in fidelity due to hardware imperfections and additional processing time. The current QMMU is rather simple due to the fact that our current quantum nodes have only have a few qubits each. Features like decoherence tracking and topology-based allocation can be part of a later version of the QMMU.

3.4 Discussion

We have discussed general design considerations for an OS for quantum network nodes, and identified the fundamental components of such an OS. We have also presented a reference architecture that abides by these requirements. Whilst our architecture is a proof of concept, and is not meant to deliver optimal performance, we implement it and test it to run simple quantum networking applications.

References

[1] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta. "Open Quantum Assembly Language". 2017. arXiv: 1707.03429.

[2] A. Dahlberg, M. Skrzypczyk, T. Coopmans, L. Wubben, F. Rozpędek, M. Pompili, A. Stolk, P. Pawełczak, R. Knegjens, J. de Oliveira Filho, R. Hanson, and S. Wehner. "A Link Layer Protocol for Quantum Networks". In: *SIGCOMM*. ACM, 2019, pp. 159–173. DOI: 10.1145/3341302.3342070.

[3] A. Dahlberg, B. van der Vecht, C. Delle Donne, M. Skrzypczyk, I. te Raa, W. Kozlowski, and S. Wehner. "NetQASM—A Low-Level Instruction Set Architecture for Hybrid Quantum–Classical Programs in a Quantum Internet". In: *Quantum Science and Technology* 7.3 (2022), p. 035023. DOI: 10.1088/2058-9565/ac753f.

[5] X. Fu, L. Riesebos, M. A. Rol, J. van Straten, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, V. Newsum, K. K. L. Loh, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudever, L. DiCarlo, and K. Bertels. "eQASM: An Executable Quantum Instruction Set Architecture". In: *HPCA*. IEEE, 2019, pp. 224–237. DOI: 10.1109/HPCA.2019.00040.

[6] X. Fu, M. A. Rol, C. C. Bultink, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudever, L. DiCarlo, and K. Bertels. "An Experimental Microarchitecture for a Superconducting Quantum Processor". In: *MICRO*. ACM, 2017, pp. 813–825. DOI: 10.1145/3123939.3123952.

[7] N. Khammassi, G. G. Guerreschi, I. Ashraf, J. W. Hogaboam, C. G. Almudever, and K. Bertels. "cQASM v1.0: Towards a Common Quantum Assembly Language ". 2018. arXiv: 1805.09607.

[8] W. Kozlowski, A. Dahlberg, and S. Wehner. "Designing a Quantum Network Protocol". In: *CoNEXT*. ACM, 2020, pp. 1–16. DOI: 10.1145/3386367.3431293.

[9] C. L. Liu and J. W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment". In: *J. ACM* 20.1 (1973), pp. 46–61. DOI: 10.1145/321738.321743.

[10] M. Pompili, C. Delle Donne, I. te Raa, B. van der Vecht, M. Skrzypczyk, G. M. 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". In: *npj Quantum Information* 8.1 (2022), p. 121. DOI: 10.1038/s41534-022-00631-2.

[11] M. Skrzypczyk and S. Wehner. “An Architecture for Meeting Quality-of-Service Requirements in Multi-User Quantum Networks”. 2021. arXiv: `2111.13124`.

[12] *NetQASM SDK*. QuTech. URL: `https://github.com/QuTech-Delft/netqasm` (visited on Feb. 28, 2023).

4 Entanglement Generation With a Quantum Networking Stack

Entanglement generation has already been demonstrated a few times by now, at various node-to-node distances, on several quantum physical platforms, and mostly on small-scale quantum networks. Scaling current quantum communication demonstrations to a large-scale quantum network will require not only advancements in quantum hardware capabilities, but also robust control of such devices to bridge the gap in user demand. Moreover, the abstraction of tasks and services offered by the quantum network should enable platform-independent applications to be executed without the knowledge of the underlying physical implementation. In this chapter, we experimentally demonstrate entanglement generation through QNodeOS and its quantum networking stack. The link layer abstracts the physical-layer entanglement attempts into a robust, platform-independent entanglement delivery service. The system is used to run full state tomography of the delivered entangled states, as well as preparation of a remote qubit state on a server by its client. Our results mark a clear transition from physics experiments to quantum communication systems, which will enable the development and testing of components of future quantum networks.

This chapter is based on the article: M. Pompili, **C. Delle Donne**, I. te Raa, B. van der Vecht, M. Skrzypczyk, G. M. 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". In: *npj Quantum Information* 8.1 (2022), p. 121. DOI: 10.1038/s41534-022-00631-2.

Contributions: M. Pompili, **C. Delle Donne**, I. te Raa, W. Kozlowski, R. Hanson and S. Wehner devised the experiment. M. Pompili, **C. Delle Donne**, I. te Raa and L. de Kluijver carried out the experiments and collected the data. M. Pompili, L. de Kluijver, A. J. Stolk and S. L. N. Hermans prepared the quantum-platform apparatus. **C. Delle Donne**, I. te Raa, B. van der Vecht, M. Skrzypczyk, G. Ferreira, P. Pawełczak and W. Kozlowski prepared the platform-independent apparatus. M. Pompili, **C. Delle Donne**, R. Hanson, and S. Wehner wrote the manuscript and the supplementary materials with input from all authors. M. Pompili, **C. Delle Donne**, I. te Raa and L. de Kluijver analyzed the data and discussed them with all authors. R. Hanson and S. Wehner supervised the research. M. Pompili and **C. Delle Donne** contributed equally to this work.

Near-term quantum networks have already yielded successful experimental results towards a future quantum internet. Fundamental primitives for entanglement-based quantum networks have been demonstrated across several physical platforms, including trapped ions [25, 37], neutral atoms [18, 32], diamond color centers [5, 19, 21, 30], and quantum dots [13, 38]. To scale up such physics experiments to intermediate-scale quantum networks, researchers have been investigating how to enclose the complex nature of quantum entanglement generation into more robust abstractions [2, 3, 4, 11, 22, 23, 29].

A common way to facilitate the scalability of complex systems is to break down their architecture into a stack of layers. Each layer in such a stack is characterized by a specific service that it provides to the high layers, reducing complexity for the higher layers, which can subsequently rely on this service. Moreover, the higher layers need no knowledge of the specific protocol and physical realization that a lower layer uses to realize the specified service. An example from classical networking is the TCP/IP stack used on the present day internet. In this stack, the link layer enables reliable transmission of data between two network nodes that are directly connected by an unreliable physical medium such as fiber or radio. Higher layers can rely on errors being detected by the link layer, and are agnostic about whether the underlying link layer protocol is Ethernet or Wi-Fi.

Several network stacks have been proposed for quantum network nodes [11, 22, 29], like the one depicted in Figure 4.1. These draw inspiration from classical architectures like the TCP/IP stack or the more generic Open System Interconnect (OSI) model. Specifically, the functional allocation of the stack proposed in Ref. [11] conceptually mirrors the TCP/IP stack in that the link layer ensures reliable (quantum) communication between adjacent nodes, and the network layer extends this service to nodes not directly connected by a physical medium themselves. We emphasize that of course no quantum data is passed up and down the layers of the stack, but only qubit metadata. Very intuitively, such metadata is similar to passing only references to an address in a physical memory up and down the stack (similar to what happens in many implementations of the TCP/IP stack in practice), while in the classical case data may of course also be copied up and down layers.

We also note that the quantum internet, and the associated quantum network stack, do not aim to replace the classical internet — they will likely coexist, as the quantum internet cannot operate without classical communication in practice. In addition to classical information used to facilitate entanglement generation, we also expect classical communication at the level of the quantum application itself (e.g. quantum key distribution), which would for practical reasons be performed using the classical internet. Finally, in a quantum network, classical communication could also be used to realize controllers like those at the core of software-defined networking (SDN) [14] to distribute information for resource scheduling and quality of service [35]. The proposed quantum network stack architecture, along with proposals for resource scheduling and routing techniques (e.g. [8, 9, 10, 17, 28, 34, 35, 40]), pave the way for larger-scale quantum networks.

In this work we experimentally demonstrate a link layer protocol for entanglement-based quantum networks. The link layer abstracts the generation of entangled states between two physically separated solid-state qubits into a robust and platform-independent service. An application can request entangled states from the link layer and then, in addition, apply local quantum operations on the entangled qubits in real-time. Using the link layer, we perform full state tomography of the generated states and achieve remote state

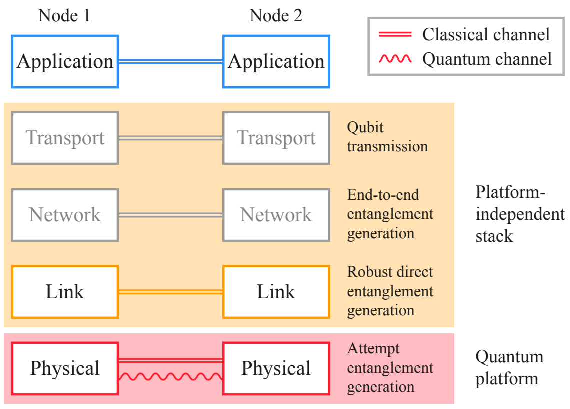


Figure 4.1: Quantum network stack architecture. At the bottom of the stack, the physical layer (red), which is highly quantum platform-dependent, is tasked with attempting entanglement generation. The link layer (yellow) uses the functionality provided by the physical layer to provide a platform-independent and robust entanglement generation service between neighboring nodes to the higher layers. Network and transport layer (not implemented in this work, grayed out) will support end-to-end connectivity and qubit transmission. Applications (blue) use the services offered by the stack to perform quantum networking tasks. Based on Ref. [11].

preparation — a building block for blind quantum computation — as well as measuring the latency of the entanglement generation service.

To evaluate correct operation and performance of our system, we measure (1) the fidelity of the generated states and (2) the latency incurred by link layer and physical layer when generating entangled pairs. For both fidelity and latency, we find that our system performs with marginal overhead with respect to previous non-platform-independent experiments. We also identify the sources of the additional overhead incurred, and propose improvements for future realizations.

4.1 Quantum Link Layer Protocol

Remote entanglement generation constitutes a fundamental building block of quantum networking. However, for a user to be able to integrate it into more complex quantum networking applications and protocols, the entanglement generation service must also be: (1) robust, meaning that the user should not have to deal with entanglement failures and retries, and that an entanglement request should result in the delivery of an entangled pair; (2) quantum platform-independent, in order for the user to be able to request entanglement without having to understand the inner workings of the underlying physical implementation; (3) on-demand, such that the user can request and consume entanglement as part of a larger quantum communication application. Robust, platform-independent, on-demand entanglement generation must figure as one of the basic services offered by a system running on a quantum network node. In other words, establishing a reliable quantum link between two directly connected nodes is the task of the first layer above the physical layer in a quantum networking protocol stack, as portrayed in Figure 4.1. Following the TCP/IP stack nomenclature, we refer to this layer as the *link layer*. We remark that, in the framework of a multi-node network, a quantum network stack should also feature a *network layer* (called *internet layer* in the TCP/IP model) to establish links between non-adjacent nodes, and optionally a *transport layer* to encapsulate qubit transmission into a service [11, 22, 29] (as shown in Figure 4.1).

Link layer service. The service provided by a link layer protocol for quantum networks should expose a few configuration parameters to its user. To ensure a platform-independent interaction with the link layer, such parameters should be common to all possible implementations of the quantum physical device. In this work, we implement a revised version of the link layer protocol proposed — but not implemented — in Ref. [11], with the following service description. The interface exposed by the link layer should allow the higher layer to specify: (1) *Remote node ID*, an identifier of the remote node to produce entanglement with (in case the requesting node has multiple neighbors); (2) *Number of entangled pairs*, to allow for the creation of several pairs with one request; (3) *Minimum fidelity*, an indication of the desired minimum fidelity for the produced pairs; (4) *Delivery type*, whether to keep the produced pair for future use (type K), measure it directly after creation (type M), or measure the local qubit immediately and instruct the remote node to keep its own for future use (type R, used for remote state preparation); (5) *Measurement basis*, the basis to use when measuring M- or R-type entangled pairs; (6) *Request timeout*, to indicate a time limit for the processing of the request. After submitting an entanglement generation request, the user should expect the link layer to coordinate with the remote node and to handle entanglement generation attempts and retries until all the desired pairs are produced (or until the timeout has expired). When completing an entanglement generation request, the link layer should then report to the above layer the following: (1) *Produced Bell state*, the result of entanglement generation; (2) *Measurement outcome*, in case of M- or R-type entanglement requests; (3) *Entanglement ID*, to uniquely identify an entangled pair consistently across source and destination of the request.

Quantum link layer protocol. A design of a quantum link layer protocol that offers the above service is the *quantum entanglement generation protocol* (QEGP) proposed by Dahlberg et al. [11]. As originally designed, this protocol relies on the underlying quantum physical layer protocol to achieve accurate timing synchronization with its remote peer and to detect inconsistencies between the local state and the state of the remote counterpart. To satisfy such requirements, QEGP is accompanied by a quantum physical layer protocol, called *midpoint heralding protocol* (MHP), designed to support QEGP on heralded entanglement-based quantum links.

Entanglement requests and agreement. QEGP exposes an interface for its user to submit *entanglement requests*. An entanglement request can specify all the aforementioned configuration parameters (remote node ID, number of entangled pairs, minimum fidelity, request type, measurement basis), and an additional set of parameters which can be used to determine the priority of the request. In the theoretical protocol proposed in Ref. [11], agreement on the requests between the nodes is achieved using a distributed queue protocol (DQP) which adds the incoming requests to a joint queue. The distributed queue, managed by the node designated as primary, ensures that both nodes schedule pending entanglement requests in the same order. Moreover, QEGP attaches a timestamp to each request in the distributed queue, so that both nodes can process the same entanglement request simultaneously.

Time synchronization. Time-scheduling entanglement generation requests is necessary for the two neighboring nodes to trigger entanglement generation at the same time, and avoid wasting entanglement attempts. QEGP relies on MHP to maintain and distribute a synchronized clock, which QEGP itself uses to schedule entanglement requests. The granularity of such a clock is only marginally important, but its consistency across the two neighboring nodes is paramount to make sure that entanglement attempts are triggered simultaneously on the two ends.

Mismatch verification. One of the main responsibilities of MHP is to verify that both nodes involved in entanglement generation are servicing the same QEGP request at the same time, which the protocol achieves by sending an auxiliary classical message to the heralding station when the physical device sends the flying qubit. The heralding station can thus verify that the messages fetched by the two MHP peers are consistent and correspond to the same QEGP request.

QEGP challenges. We identify three main challenges that would be faced when deploying QEGP on a large-scale quantum network, while suggesting an alternative solution for each of these. (C1) Using a link-local protocol (DQP) to schedule entanglement requests, albeit sufficient for a single-link network, becomes challenging in larger networks, given that a node might be connected to more than just one peer. In such scenarios, the scheduling of entanglement requests can instead be deferred to a centralized scheduling entity, one which has more comprehensive knowledge of the entire (sub)network [35]. (C2) Entrusting the triggering of entanglement attempts to QEGP would impose very stringent real-time constraints on the system where QEGP itself is deployed — even microsecond-level latencies on either side of the link can result in out-of-sync (thus wasteful) entanglement attempts. While Dahlberg et al. [11] identify this problem as well, the original MHP protocol assumes that both QEGP peers issue an entanglement command to the physical layer at the same clock cycle. In this scheme, MHP initiates an entanglement attempt regardless of the state of the remote counterpart. We believe that fine-grained entanglement attempt synchronization should pertain to the physical layer only, building on the assumption that the real-time controllers deployed at the physical layer of each node are anyway highly synchronized [30]. (C3) Checking for request mismatches at the heralding station requires the latter to be capable of performing such checks in real-time. Given that the two neighboring MHP protocols have to anyway synchronize before attempting entanglement, we suggest that, as an alternative approach, consistency checks be performed at the nodes themselves, rather than at the heralding station, just before entering the entanglement attempt routine.

4.2 Revised Protocol

To address the present QEGP and MHP challenges with the proposed solutions, we have made some modifications to the original design of the two protocols. In particular, we adopted a centralized request scheduling mechanism [35] to tackle challenge (C1), we delegated the ultimate triggering of entanglement attempts to MHP as a solution to challenge (C2), and we assigned request mismatch verification to the MHP protocol running on each node, rather than to the heralding station, to address challenge (C3).

Centralized request scheduling. To avoid using a link-local protocol (DQP) to schedule entanglement requests, our version of QEGP defers request scheduling to a *centralized request scheduler*, whereby a node's entanglement generation schedule is computed on the basis of the whole network's needs. Delegating network scheduling jobs to centralized entities is, albeit not the only alternative, a common paradigm of classical networks, and especially of software-defined networking (SDN) — a concept that has been recently investigated in the context of quantum networking [2, 23]. In large networks, such controllers are *logically centralized*, but *physically distributed*, to ensure their reliability and availability in spite of possible failures. In our system, the centralized scheduler produces a time-division multiple access (TDMA) network schedule — one for each node in the network — where each time bin is reserved for a certain class of entanglement generation requests [35]. A class of requests may comprise, for instance, all requests coming from the same application and asking for the same fidelity of the entangled states. While reserving time bins may be redundant in a single-link network, integrating a centralized scheduling mechanism early on into the link layer protocol will facilitate future developments.

MHP synchronization and timeout. Although centralized request scheduling makes the synchronization of QEGP peers easier, precise triggering of entanglement attempts should still be entrusted to the component of the system where time is the most deterministic — in our case, the physical layer protocol MHP. In contrast to Ref. [11], once MHP fetches an entanglement instruction from QEGP, the protocol announces itself as ready to its remote peer, and waits for the latter to do so as well. After this synchronization step succeeds, the two MHP peers can instruct the underlying hardware to trigger an entanglement attempt at a precise point in time. If, instead, one of the two MHP peers does not receive announcements from its remote counterpart within a set timeout, it can conclude that the latter is not ready, or temporarily not responsive, and can thus return control to QEGP without wasting entanglement attempts. This MHP synchronization step is also useful for the two sides to verify that they are processing the same QEGP request, and thus catch mismatches.

The MHP synchronization routine inherently incurs some overhead, which is also larger on longer links. We mitigate this overhead by batching entanglement attempts — that is, the physical layer attempts entanglement multiple times after synchronization before reporting back to the link layer. The maximum number of attempts per batch is a purely physical-layer parameter, and it has no relation with the link layer entanglement request timeout parameter described in Ref. [11] — although batches should be small enough for the link layer timeout to make sense.

The original design of the QEGP and MHP protocols, as well as our revision, specifies the conceptual interaction between the two protocols and the service exposed to a higher layer in the system, but does not impose particular constraints on how to implement link layer and physical layer, how to realize the physical interface between them, and how to configure things such as the centralized request scheduler and the entanglement attempt procedure. Figure 4.2 gives an overview of the architecture of our quantum network nodes. We briefly describe our most relevant implementation choices here and in the physical layer section.

Application processing. At the application layer, user programs — written in Python using a dedicated *software development kit* [45] — are processed by a rudimentary com-

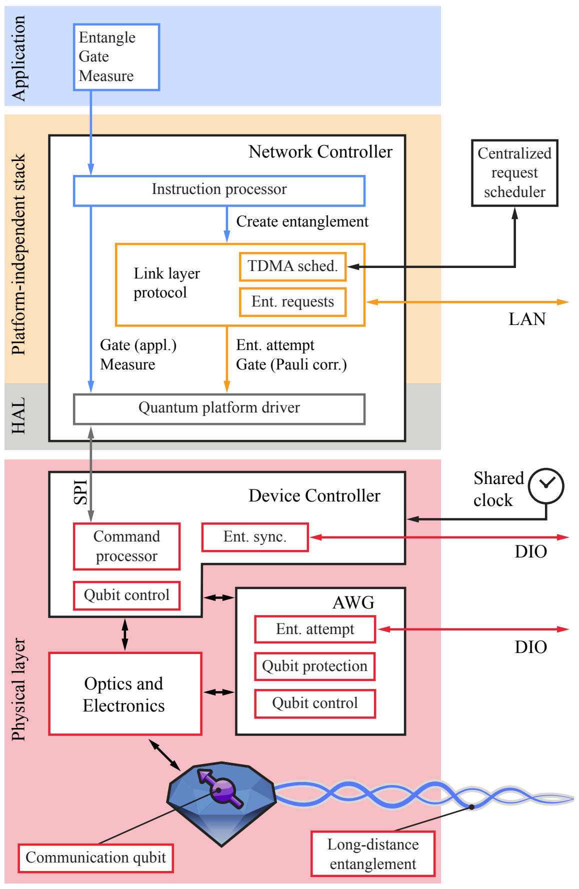


Figure 4.2: Quantum network node architecture. From top to bottom: At the application layer, a simple platform-independent routine is sent to the network controller. The network controller implements the platform-independent stack — in this work only the link layer protocol — and a hardware abstraction layer (HAL) to interface with the physical layer's device controller. An instruction processor dispatches instructions either directly to the physical layer, or to the link layer protocol in case a remote entangled state is requested by the application. The link layer schedules entanglement requests and synchronizes with the remote node (on a local area network) using a time-division multiple access (TDMA) schedule computed by a centralized scheduler (external). At the physical layer, the device controller fetches commands from — and replies with outcomes to — the network controller. Driven by a clock shared with the neighboring node, it performs hard-real-time synchronization for entanglement generation using a digital input/output (DIO) interface. By controlling the optical and electronic components (among which an arbitrary waveform generator, AWG), the device controller can perform universal quantum control of the communication qubit in real-time, as well as attempt long-distance entanglement generation with the neighboring node.

pilation stage, which translates abstract quantum networking applications into gates and operations supported by our specific quantum physical platform. Such gates and operations are expressed in a low-level assembly-like language for quantum networking applications called *NetQASM* [12]. As part of our software stack, we also include an *instruction processor*, conceptually placed above the link layer, which is in charge of dispatching entanglement requests to QEGP and other application instructions to the physical layer directly.

Interface. Ref. [11] did not provide a specification of the interface to be exposed by the physical layer. We designed this interface such that the physical layer can accept *commands* from the higher layer, specifically: (1) qubit initialization (`INI`), (2) qubit measurement (`MSR`), (3) single-qubit gate (`SQG`), (4) entanglement attempt (`ENT`, or `ENM` for M- or R-type requests), (5) pre-measurement gates selection (`PMG`, to specify in which basis to measure the qubit for M- or R-type requests). For each command, the physical layer reports back an *outcome*, which indicates whether the command was executed correctly, and can bear the result of a qubit measurement and the Bell state produced after a successful entanglement attempt. Our software stack also comprises a *hardware abstraction layer* (HAL) that sits below QEGP and the instruction processor. The HAL encodes and serializes commands and outcomes, and is thus used to interface with the device controller.

TDMA network schedule. Designing a full-blown centralized request scheduler is a challenge in and of its own, outside the scope of this work. Instead of implementing such a scheduler, we compute static TDMA network schedules [35] and install them manually on the two network nodes upon initialization. TDMA network schedules are redundant in a one-link network. Time-binning network activity also forces nodes to only process entanglement requests at the beginning of a time division, thus introducing latency and idle time. Particularly, longer time bins potentially result in entanglement requests to wait longer to be processed. However, an application asking for multiple entangled pairs with just one request would experience smaller average latencies, as all pairs—but the first one—would be generated in close succession.

TDMA schedules for our simple single-link experiments are quite trivial, as the network resources of a node are not contended by multiple links. In particular, schedules are just a constant division of 20 ms time bins, each of which is reserved to the only application running. We chose the duration of the time-bin — somewhat arbitrarily, given the small effect on our experiments — to be equal to 1000 communication cycles between the device controller and the network controller ($20\,\text{ms} = 1000 \times 20\,\mu\text{s}$).

Entanglement attempts. Producing entanglement on a link can take several attempts. To minimize the number of `ENT` commands fetched by MHP from QEGP, as well as to mitigate the MHP synchronization overhead incurred after each entanglement command, we batch entanglement attempts at the MHP layer, such that synchronization and outcome reporting only happens once per batch of attempts.

Delivered entangled states. In our first iteration, we implemented QEGP such that it always delivers $|\Phi^+\rangle$ Bell states to the higher layer. This means that, when the physical layer produces a different Bell state, QEGP (on the node where the entanglement request originates) issues a single-qubit gate — a Pauli correction — to transform the entangled pair into the $|\Phi^+\rangle$ state (we abbreviate the four two-qubit maximally entangled Bell states as $|\Phi^\pm\rangle = (|00\rangle \pm |11\rangle)/\sqrt{2}$ and $|\Psi^\pm\rangle = (|01\rangle \pm |10\rangle)/\sqrt{2}$). A future version of QEGP could allow the user to request any Bell state, and could extract the Pauli correction from QEGP so that the application itself can decide, depending on the use case, whether to apply the correction or not.

Mismatch verification. As per our design specification, MHP should also be responsible for verifying that the entanglement commands coming from the two QEGP peers belong to the same request. We did not implement this feature yet because, in our simple quantum network, we do not expect losses on the classical channel used by the two MHP parties to communicate — a lossy classical channel would be the primary source of inconsistencies at the MHP layer [11]. However, we believe that this verification step will prove very useful in real-world networks where classical channels do not behave as predictably.

Deployment. We implemented QEGP as a software module in a system that also includes the instruction processor and the hardware abstraction layer. QEGP, the instruction processor and the hardware abstraction layer, forming the *network controller*, are implemented as a C/C++ standalone runtime developed on top of FreeRTOS, a real-time operating system for embedded platforms [44]. The runtime and the underlying operating system are deployed on a dedicated Avnet MicroZed — an off-the-shelf platform based on the Zynq-7000 SoC, which hosts two ARM Cortex-A9 processing cores, of which only one is used, clocked at 667 MHz. QEGP connects to its remote peer via TCP over a Gigabit Ethernet interface. The interface to the physical layer is realized through a 12.5 MHz SPI connection. The user application is sent from a general-purpose 4-core desktop machine running Linux, which connects to the instruction processor through the same Gigabit Ethernet interface that QEGP uses to communicate with its peer.

4.3 Physical Layer Control in Real-Time

In this section, we outline the design and operation of the physical layer, which executes the commands issued by the higher layers on the quantum hardware and handles time-critical synchronization between the quantum network nodes. The physical layer of a quantum network, as opposed to the apparatus of a physics experiment, needs to be able to execute commands coming from the layer above in real-time. Additionally, when performing the requested operations, it needs to leave the quantum device in a state that is compatible with future commands (for example, as discussed below, it should protect qubits from decoherence while it awaits further instructions). Finally, if a request cannot be met (e.g. the local quantum hardware is not ready, the remote quantum hardware is not available, etc.), the physical layer should notify the link layer of the issue without interrupting its service.

Our quantum network is composed of two independent nodes based on diamond NV centers physically separated by ≈2 m (see Figure 4.2 for the architecture of one node). We will refer to the two nodes as *client* and *server*, noting that this is only a logical separation useful to describe the case studies — the two nodes have the exact same capabilities. On each node, we implement the logic of the physical layer in a state-machine-based algorithm deployed on a time-deterministic microcontroller, the *device controller* (Jäger ADwin Pro II, based on Zynq-7000 SoC, dual-core ARM Cortex-A9, clocked at 1 GHz). Additionally, each node uses an arbitrary waveform generator (AWG, Zurich Instruments HDAWG8, 2.4 GSa/s, 300 MHz sequencer) for nanosecond-resolution tasks, such as fast optical and electrical pulses; the use of such a user-programmable FPGA-based AWG, as opposed to a more traditional upload-and-play instrument (such as the ones used in Ref. [30]), enables the real-time control of our quantum device.

Single node operation. On our quantum platform, before a node is available to execute commands, it needs to perform a qubit readiness procedure called *charge and resonance check* (CR check). This ensures that the qubit system is in the correct charge state and that the necessary lasers are resonant with their respective optical transitions. Other quantum platforms might have a similar preparation step, such as loading and cooling for atoms and ions [32, 37]. Once the CR check is successful, the device controller can fetch a command from the network controller. Depending on the nature of the command, the device controller might need to coordinate with other equipment in the node or synchronize with the device controller of the other node.

For qubit initialization and measurement commands (`INI` and `MSR`), the device controller shines the appropriate laser for a pre-defined duration (`INI`≈100 µs, `MSR`≈10 µs). Both operations are deterministic and carried out entirely by the device controller.

Single qubit gates (`SQG`) require the coordination of the device controller and the AWG. For our communication qubits, they consist of generating an electrical pulse with the AWG (duration ≈100 ns), which is then multiplied to the qubit frequency (≈2 GHz), amplified and finally delivered to the quantum device. The link layer can request rotations in steps of $\pi/16$ around the X, Y or Z axis of the Bloch sphere (here we implement only X and Y rotations, Z rotations will be implemented in the near future, see Appendix C). When a new gate is requested by the link layer, the device controller at the physical layer informs the AWG of the gate request via a parallel 32-bit DIO interface. The AWG will then select one of the 64 pre-compiled waveforms, play it, and notify the device controller that the gate has been executed. The device controller will in turn notify the network controller of the successful operation.

After the rotation has been performed, our qubit — if left idling — would lose coherence in ≈5 µs. A coherence time exceeding 1 s has been reported on our platform [1] using decoupling sequences (periodic rotations of the qubit that shield it from environmental noise). By interleaving decoupling sequences and gates, one can perform extended quantum computations [6]. These long sequences of pulses have in the past been calculated and optimized offline (on a PC), then uploaded to an AWG, and finally executed on the quantum devices with minimal interaction capabilities (mostly binary branching trees, see [30]). In our case, it is impossible to pre-calculate these sequences, since we cannot know in advance which gates are going to be requested by the link layer. To solve this challenge, we implement a qubit protection module on the AWG, that interleaves decoupling sequences with the requested gates in real-time. As soon as the first gate in a sequence is requested, the AWG starts a decoupling sequence on the qubit. Then, it periodically checks if a new gate has been requested, and if so, it plays it at the right time in the decoupling sequence. The AWG will continue the qubit protection routine until the device controller will ask for it to stop (e.g. to perform a measurement). This technique allows us to execute universal qubit control without prior knowledge of the sequence to be played, and — crucially — in real-time.

Entanglement generation. Differently from the commands previously discussed, attempting entanglement generation (`ENT`) requires tight timing synchronization between the device controllers — and AWGs — of the two nodes. In our implementation, the two device controllers share a common 1 MHz clock as well as a DIO connection to exchange synchronization messages (see Ref. [30]). When the device controllers are booted, they synchronize an internal cycle counter that is used for time-keeping, and is shared, at each node, with their respective network controllers to provide timing information to the link layer and the higher layers. Over larger distances, one could use well-established protocols to achieve sub-nanosecond, synchronized, GPS-disciplined common clocks [46].

When a device controller fetches an ENT command, it starts a three-way handshake procedure with the device controller of the other node. If the other node has also fetched an ENT command, they will synchronize and proceed with the entanglement generation procedure. If one of the two nodes is not available (e.g. it is still trying to pass the CR check) the other node will time out, after 0.5 ms, and return an *entanglement synchronization failure* (ENT_SYNC_FAIL) to its link layer. The duration of the timeout is chosen such that is comparable with the average time taken by a node to pass the charge and resonance check (if correctly on resonance). This is to avoid unnecessary interactions between physical layer and link layer. After the entanglement synchronization step, the device controllers proceed with an optical phase stabilization cycle [30], and then the AWGs are triggered to attempt entanglement generation. In our implementation, one device controller (the server's) triggers both AWGs to achieve sub-nanosecond jitter between the two AWGs (see Appendix C for a discussion on longer distance implementation). Each entanglement attempt lasts 3.8 μs, and includes fast qubit initialization, communication-qubit to flying-qubit entanglement, and probabilistic entanglement swapping of the flying qubits [30]. The AWGs attempt entanglement up to 1000 times before timing out and reporting an entanglement failure (ENT_FAIL). Longer batches of entanglement attempts would increase the probability that one of the nodes goes into the unwanted charge state (and therefore cannot produce entanglement, see Appendix C). While in principle possible, we did not implement, in this first realization, the charge stabilization mechanism proposed in Ref. [19] that would allow for significantly longer batches of entanglement attempts.

If an entanglement generation attempt is successful (probability $\approx 5 \times 10^{-5}$), the communication qubits of the two nodes will be projected into an entangled state (either $|\Psi^+\rangle$ or $|\Psi^-\rangle$, depending on which detector clicked at the heralding station). To herald success of the entanglement attempt, a CPLD (Complex Programmable Logic Device, Altera MAX V 5M570ZF256C5N) sends a fast digital signal to both AWGs and device controllers, to prevent a new entanglement attempt from being played (which would destroy the generated entangled state). When the heralding signal is detected, the AWGs enter the qubit protection routine and wait for further instructions from the device controllers, which in turn notify the link layer of the successful entanglement generation, as well as which state was generated.

To satisfy M- or R-type entanglement requests, the link layer can instruct the physical layer to apply an immediate measurement to the entangled qubit by means of an ENM command. Up until heralding of the entangled state, the physical layer operates as it does for the ENT command. When the state is ready, it proceeds immediately with a sequence of single qubit gates (as prescribed by an earlier PMG command) and a qubit measurement. The result of the measurement, together with which entangled state was generated, is communicated to the link layer. It is worth noting that the two nodes could fetch different types of requests and still generate entanglement. In fact, this will be used later in the remote state preparation application.

4.4 Evaluation

To demonstrate and benchmark the capabilities of the link layer protocol, the physical layer, and of our system as a whole, we execute — on our two-node network — three quantum networking applications, all having a similar structure: the client asks for an entangled pair with the server, which QEGP delivers in the $|\Phi^+\rangle$ Bell state, and then both client and server measure their end of the pair in a certain basis. First, we perform full quantum state tomography of the delivered entangled states. Second, we request and characterize entangled states of varying fidelity. Third, we execute remote preparation of qubit states on the server by the client. For all three applications, we study the quality of the entangled pairs delivered by our system. Additionally, we use the second application to assess the latency incurred by our link layer, and to compare it to the overall entanglement generation latency, including that of the physical layer. Crucially, the three applications are executed back-to-back on the quantum network, without any software or hardware changes to the system — the only difference being the quantum-platform-independent application sent to the instruction processor.

The sequence diagram in Figure 4.3a exemplifies the general flow between system components during the execution of an application. At first, the instruction processor issues a request to create entanglement to link layer (`CREATE`). Then, the client's link layer forwards the request with the server's counterpart (Forward `CREATE`). The request is processed as soon as the designated time bin in the TDMA schedule starts, at which point the first entanglement command (`ENT`) is fetched by physical layer. After an entangled state is produced successfully (`PSI_PLUS`), the link layer of the client issues, if needed, a Pauli correction (π rotation around the X axis, `SQG X180`) to deliver the pair in the $|\Phi^+\rangle$ state. Finally, the instruction processor issues a gate ($\pi/2$ rotation around the X axis, `SQG X90`) and a measurement (`MSR`) to read out the entangled qubit in a certain basis, and receives an outcome from the physical layer (`0`). Figure 4.3b illustrates the actual latencies between these interactions in one iteration of the full state tomography application.

For all our experiments, we configured TDMA time bins to be of 20 ms. In a larger network, the duration of time bins should be calibrated according to the average time it takes, on a certain link, to produce an entangled pair of a certain fidelity [35]. By doing so, one can maximize network usage and thus reduce qubit decoherence on longer end-to-end paths. However, in our single-link network, the duration of time bins only influences the frequency at which new entanglement requests are processed. Our time bin duration accommodates up to four batches of 1000 entanglement attempts.

Full quantum state tomography. The first application consists in generating entangled states at the highest *minimum fidelity* currently available on our physical setup (0.80), and measuring the two entangled qubits in varying bases to learn their joint quantum state. We measure all 9 two-node correlators ($\langle XX\rangle, \langle XY\rangle, \ldots, \langle ZZ\rangle$) as well as all their $\pm$ variations ($\langle +X+X\rangle, \langle +X-X\rangle$, etc.) to minimize the bias due to measurement errors. For each of the $9 \times 4 = 36$ combinations, we measure 125 data points, for a total of 4500 entangled states generated and measured.

The collected measurement outcomes are then analyzed using QInfer [16], in particular the Monte Carlo method described in Ref. [15] for Bayesian estimation of density matrices from tomographic measurements. The reconstructed density matrix is displayed

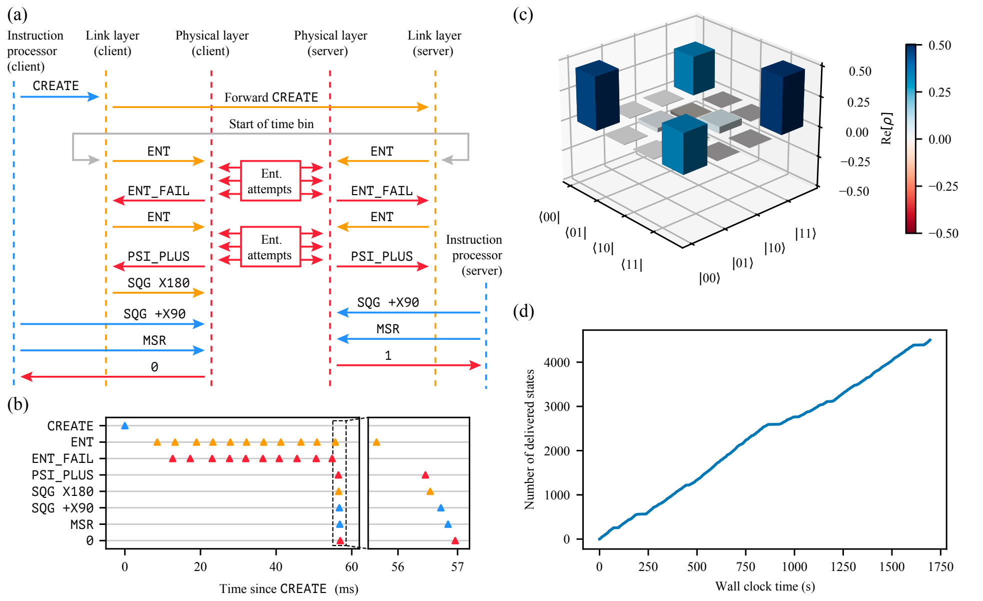


Figure 4.3: Full state tomography with the quantum network stack. (a) Sequence diagram of the communication steps across the network stack and the two nodes to perform one repetition of the tomography application (in particular, measurement of the $\langle YY\rangle$ correlator). The coloring follows that of Figure 4.1. CREATE: entanglement request, ENT: entanglement attempts request, ENT_FAIL: failed the batch of entanglement attempts, PSI_PLUS: successful entanglement attempt with generated state Ψ^+, SQG: single-qubit gate, X180: 180° rotation around X axis, MSR: qubit measurement, 0/1: qubit measurement outcome. Note that the client's link layer protocol requests a X180 gate after entanglement generation to deliver the $|\Phi^+\rangle$ Bell state to the higher layer. (b) Example time trace of (a) for the client. Several batches of entanglement attempts are required before an entangled state is heralded. On the right, a zoomed-in part of the trace (corresponding to the dashed box in the left plot). (c) Reconstructed density matrix of the states delivered by the link layer. Only the real part is plotted (imaginary elements are all ≈0, see main text). We estimate a fidelity F with $|\Phi^+\rangle$ of $F = 0.783(7)$. (d) Total number of delivered states over time. The occasional pauses in entanglement delivery (plateaus) are due to the client's NV center becoming off-resonant with the relevant lasers (see Appendix C). Differences in slope are due to changes in resonance conditions that increase the time necessary to pass the charge and resonance check.

in Figure 4.3c (only the real part is shown) and its values and uncertainties are

$$\mathrm{Re}[\rho] = \begin{pmatrix} 0.442(6) & 0.003(3) & 0.003(2) & 0.328(5) \\ 0.003(3) & 0.033(6) & -0.023(5) & -0.000(5) \\ 0.003(2) & -0.023(5) & 0.056(4) & -0.003(4) \\ 0.328(5) & -0.000(5) & -0.003(4) & 0.469(7) \end{pmatrix},$$

$$\mathrm{Im}[\rho] = \begin{pmatrix} 0 & -0.014(3) & -0.005(7) & 0.032(5) \\ 0.014(3) & 0 & -0.002(4) & 0.001(5) \\ 0.005(7) & 0.002(4) & 0 & -0.000(7) \\ -0.032(5) & -0.001(5) & 0.000(7) & 0 \end{pmatrix}.$$

Here $\rho_{ij,mn} = \langle ij|\,\rho\,|mn\rangle$, with i, m (j, n) being the client (server) qubit states in the computational basis. The uncertainty on each element of the density matrix is calculated as the standard deviation of that element over the probability distribution approximated by the Monte Carlo reconstruction algorithm (probability distribution approximated by 1×10^5 Monte Carlo particles [15]). It is then possible to estimate the fidelity of the delivered entangled states with respect to the maximally entangled Bell state, which we find to be $F = 0.783(7)$. The measured fidelity is slightly lower (≈3 %) than what measured in Ref. [30] without the use of the QEGP abstraction (and the whole network controller where QEGP runs). This discrepancy could be due to the additional physical-layer decoupling sequences required for real-time operation (≈300 µs) and the additional single-qubit gate issued by the link layer to always deliver $|\Phi^+\rangle$ (the physical layer can produce either $|\Psi^+\rangle = (|01\rangle + |10\rangle)/\sqrt{2}$ or $|\Psi^-\rangle = (|01\rangle - |1\rangle)/\sqrt{2}$, see Refs. [19, 30]).

It is to be noted that, in order to obtain the most faithful estimate of the generated state (see Section 4.5 for details), the measured expectation values are corrected, in post-processing, to remove known tomography errors of both client and server [26], and events in which at least one physical device was in the incorrect charge state.

Finally, we show, in Figure 4.3d, that our system can sustain a fairly stable entanglement delivery rate over ≈30 min of data acquisition — plateaus and changes in slope can be attributed to varying conditions of resonance between the NV centers and the relevant lasers (see Appendix C).

Latency VS fidelity. The QEGP interface allows its user to request entangled pairs at various minimum fidelities. For physical reasons, higher fidelities will result in lower entanglement generation rates [19, 38]. The trade-off between fidelity and throughput is particularly interesting in a scenario where some applications might require high-fidelity entangled pairs and are willing to wait a longer time, while others might prefer lower-fidelity states but higher rates [11]. Clearly, for the link layer to offer a range of fidelities to choose from, the underlying physical layer must support such a range. We benchmark the capabilities of the link layer and of the physical layer to deliver states at various fidelities in a single application by measuring the $\langle XX\rangle$, $\langle YY\rangle$ and $\langle ZZ\rangle$ correlators (and their $\pm$ variations, as we did above, for a total of $3 \times 4 = 12$ correlators) for seven different target fidelities, (0.50, 0.55, 0.60, 0.65, 0.70, 0.75, 0.80). We generate 1500 entangled states per fidelity, for a total of 10 500 delivered states. With this case study, we analyze both the resulting fidelity and the system's latency for different requested fidelities.

The results for measured fidelity versus requested fidelity are shown in Figure 4.4a. It is worth noting that the application iterates over the range of fidelities in real-time, and thus the physical layer is prepared to deliver any of them at any point. We calibrate the physical layer to deliver states of slightly higher fidelity than the requested ones (0.03 more), since entanglement requests specify the *minimum* desired fidelity. The measured fidelities are — within measurement uncertainty — always matching or exceeding the requested minimum ones (the dashed gray line in Figure 4.4a is the $y = x$ diagonal). As in the previous application, measurement outcomes are post-processed to eliminate tomography errors and events in which the physical devices were in the incorrect charge state (we refer to the latter as *charge state correction*). For arbitrary applications that use the delivered entangled states for something other than statistical measurements, applying the second correction directly at the link layer might prove challenging, since the information concerning whether to discard an entangled pair is only available at the physical layer *after* the entangled state is delivered to the link layer (when the next CR check is

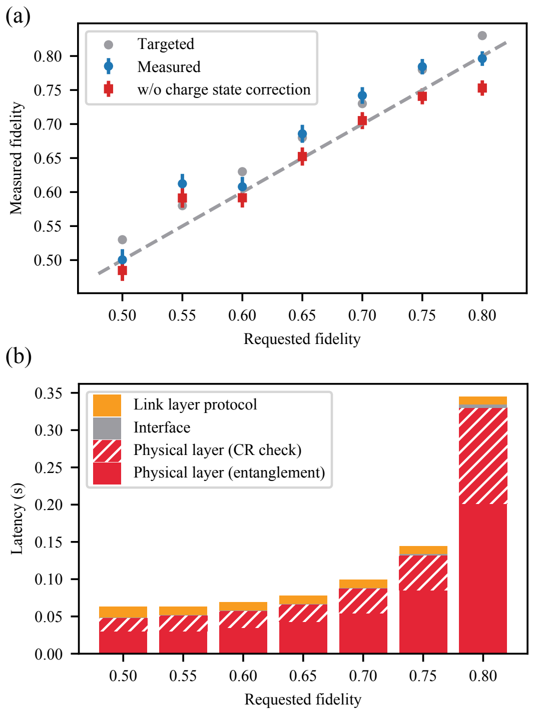


Figure 4.4: Performance of the entanglement delivery service. (a) Measured fidelity of the states delivered by the link layer for varying requested fidelity. Targeted fidelity at the physical layer is 0.03 higher than the link layer protocol's *minimum fidelity* request. When not correcting for wrong charge state events, fidelity is reduced by a few percents (see Section 4.5). Error bars represent 1 s.d. (b) Average latency of the entanglement delivery per requested fidelity, broken down into sources of latency. Entanglement generation and charge and resonance check at the physical layer are the largest sources of latency (at higher fidelities, more entanglement attempts are required before success). Running the link layer protocol introduces a small but measurable overhead (≈10 ms) to the entanglement generation procedure, which does not depend on the requested fidelity, and that could be mitigated by requesting multiple entangled states in a single instruction. The communication delays between quantum network controller and quantum device controller (Interface) introduce negligible overall latency.

performed). However, a mechanism to identify *bad* entangled pairs retroactively at the link layer — like the expiry functionality included in the original design of QEGP [11] — could be used to discard entangled states after they have been delivered by the physical layer. For completeness, we also report, again in Figure 4.4a, the measured fidelity when the wrong charge state correction is not applied.

For each requested fidelity we also measure the entanglement generation latency [11], defined as the time between the issuing of the `CREATE` request to the link layer, until the successful entanglement outcome reported by the physical layer (refer to Figure 4.3a for a diagram of the events in between these two). Figure 4.4b shows the measured average latency, grouped by requested fidelity and broken down into the various sources of latency. When calculating the average latencies, we have ignored entanglement requests that required more than 10 s to be fulfilled. These high-latency requests correspond to

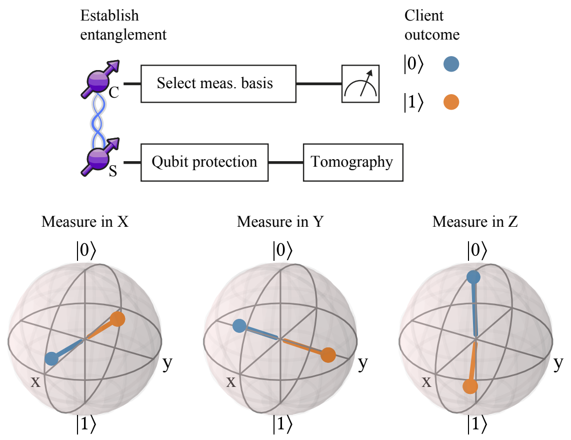


Figure 4.5: Tomography of states prepared on the server by the remote client. For each chosen measurement axis of the client (X, Y, Z), and for each obtained measurement outcome at the client (|0⟩, |1⟩), a different state is prepared on the server. Plotted on the Bloch spheres are the results of the tomography on the server's qubit. Uncertainties on each coordinate are ≈0.05 (see Section 4.5). We find an average fidelity of $F = 0.853(8)$.

the horizontal plateaus of Figure 4.3d. The main contribution to the total latency comes from the entanglement generation process at the physical layer, followed by the NV center preparation time (CR check). Both latency values are consistent with the expected number of entanglement attempts required by the single-photon entanglement protocol employed at the physical layer [19]. The link layer protocol adds, on average, ≈10 ms of extra latency to all requests, regardless of their fidelity. This is due partly to the synchronization of the `CREATE` request between the two nodes (i.e. a simple TCP message), but mostly to the nodes having to wait for the next time bin in the network schedule to start. We remark that, by requesting multiple entangled states in a single `CREATE`, one can distribute this overhead over many generated pairs, to the point where it becomes negligible. While our applications did not issue multi-pair `CREATE` requests, this would be the more natural choice for real applications, and would result in better utilization of the allocated time bins. Finally, the overhead incurred by the interface between microcontrollers is rather small (barely visible in Figure 4.4b), but could however be further reduced by integrating device controller and network controller into a single device. It is worth mentioning that, in our simple scenario in which each entanglement request is only submitted to QEGP after the previous one completes, and thus the request queue never grows larger than one element, throughput happens to be almost exactly the same as the inverse of latency, and hence it is not reported here.

Overall, we observe that the extra entanglement generation latency incurred when deploying an abstraction layer (QEGP) on top of the physical layer, while not too modest, is only a small part of the whole, particularly at higher fidelities. Nevertheless, optimizing the length of TDMA time bins could result in an even smaller overhead.

Remote state preparation. One of the use cases of the QEGP service is to prepare quantum states on a remote server [11]. Remote state preparation is a fundamental step to execute a blind quantum computation application [7], whereby a client quantum computer with limited resources can run applications on a powerful remote quantum server using the many qubits the server has, while keeping the performed computation private.

Remote state preparation is different from the previous two cases in that the client can measure its end of the entangled pair as soon as the pair is generated, while the server has to keep its qubit alive waiting for further instructions. For such a scenario, the client can make use of QEGP's service to issue R-type entanglement requests, so that the local end of the entangled pair can be measured (in a certain basis) as soon as it is generated, while the server's qubit can be protected for later usage. An R-type entanglement request results in an `ENM` command on the client and an `ENT` command on the server. For this type of requests (as well as for M-type ones), since the local end of the pair is measured immediately, the client's QEGP can skip the Pauli correction used to always deliver $|\Phi^+\rangle$, and can instead apply a classical correction to the received measurement outcome.

To showcase this feature of QEGP we use the client node to prepare the six cardinal states on the server ($|\pm x\rangle$, $|\pm y\rangle$, $|0\rangle$ and $|1\rangle$) by having the client measure its share of the entangled state in the six cardinal bases. We then let the server measure the prepared states — again in the six cardinal bases — to perform tomography. For each client measurement basis, and for each server tomography basis, we deliver 125 entangled states at a requested fidelity of 0.80, for a total of $6 \times 6 \times 125 = 4500$ remote state preparations. The results are presented in Figure 4.5, which displays the tomography of the prepared states on the server, for the three different measurement axes of the client and the two possible measurement outcomes of the client. The prepared states are affected by the measurement error of the client ($F_0 = 0.928(3)$, $F_1 = 0.997(1)$): an error in the measurement of the client's qubit results in an incorrect identification of the state prepared on the server. By alternating between positive and negative readout orientations, we make sure that the errors affect all prepared states equally, instead of biasing the result. We note that we exclude, once again, events in which at least one of the two devices was in the wrong charge state, and we correct for the known tomography error on the server (results without corrections are in Section 4.5). Overall, we find an average remote state preparation fidelity of $F = 0.853(8)$. The asymmetry in the fidelity of the $|0\rangle$ and $|1\rangle$ states is caused by the asymmetry in the populations $\langle 01|\,\rho\,|01\rangle$ vs $\langle 10|\,\rho\,|10\rangle$ of the delivered entangled state, which in turn is due to the double $|0\rangle$ occupancy error of the single-photon protocol used to generate entanglement [19, 30].

4.5 Results With and Without Corrections

The data presented in the main text is corrected for known measurement errors, and events in which at least one of the two devices was in the wrong charge state are removed (the CR check following the delivery of entanglement reports zero counts). While it is useful to correct for such errors in order to obtain the most faithful reconstruction of the delivered states, these errors cannot always be avoided in a real network scenario. For completeness, we report here the results first without any corrections applied, and then with only the measurement error correction applied. All the results, the raw datasets, and the software to analyze them, are available at Ref. [43].

Full quantum state tomography. The events in which the two devices generated 0 photon counts in the following CR check were 37 for the client and 380 for the server (out of the 4500 total). When combined, (client or server in the wrong charge state), we obtain 417 events (in zero events both client and server were in the wrong charge state). Without any corrections (tomography errors or wrong charge state), we obtain the following density matrix (which has a fidelity with the target Bell state F=0.681(16)):

$$\mathrm{Re}[\rho] = \begin{pmatrix} 0.397(9) & 0.011(9) & 0.001(7) & 0.256(14) \\ 0.011(9) & 0.058(14) & -0.005(13) & -0.007(9) \\ 0.001(7) & -0.005(13) & 0.092(12) & -0.027(13) \\ 0.256(14) & -0.007(9) & -0.027(13) & 0.452(9) \end{pmatrix},$$

$$\mathrm{Im}[\rho] = \begin{pmatrix} 0 & 0.000(18) & -0.029(9) & 0.036(9) \\ -0.000(18) & 0 & 0.010(12) & -0.002(8) \\ 0.029(9) & -0.010(12) & 0 & -0.000(8) \\ -0.036(9) & 0.002(8) & 0.000(8) & 0 \end{pmatrix}$$

Only applying tomography error correction (but not removal of wrong charge state events) yields the following density matrix (fidelity F=0.744(11)):

$$\mathrm{Re}[\rho] = \begin{pmatrix} 0.421(7) & -0.001(4) & -0.013(5) & 0.300(8) \\ -0.001(4) & 0.022(8) & -0.020(6) & -0.021(7) \\ -0.013(5) & -0.020(6) & 0.091(5) & -0.015(5) \\ 0.300(8) & -0.021(7) & -0.015(5) & 0.466(5) \end{pmatrix}$$

$$\mathrm{Im}[\rho] = \begin{pmatrix} 0 & 0.004(4) & -0.018(3) & 0.032(6) \\ -0.004(4) & 0 & 0.021(6) & 0.002(5) \\ 0.018(3) & -0.021(6) & 0 & 0.002(5) \\ -0.032(6) & -0.002(5) & -0.002(5) & 0 \end{pmatrix}$$

Fidelity VS rate. The events in which the two devices generated 0 photon counts in the following CR check were 74 for the client and 709 for the server (out of the 10 500 total). When combined, (client or server in the wrong charge state), we obtain 781 events (there were two events in which both client and server were in the wrong charge state). Without any corrections (tomography errors or wrong charge state), we obtain the following delivered fidelities: 0.454(18), 0.540(18), 0.548(17), 0.596(17), 0.640(16), 0.674(16), 0.679(15). Only applying tomography error correction (but not removal of wrong charge state events) yields the following fidelities: 0.485(15), 0.591(14), 0.592(14), 0.652(13), 0.705(13), 0.741(12), 0.753(11).

Remote state preparation. As mentioned in the main text, for the remote state preparation analysis, we only apply the tomography error correction for the server, while remove wrong charge state events of both the server and the client. The events in which the two devices generated 0 photon counts in the following CR check were 29 for the client and 365 for the server (out of the 4500 total). When combined, (client or server in the wrong charge state), we obtain 394 events (there were zero events in which both client and server were in the wrong charge state). Following are the numerical values that result in the plot in the main text (average fidelity F=0.853(8)):

Client	Server			
	⟨X⟩	⟨Y⟩	⟨Z⟩	Fidelity
Measured $\|+X\rangle$	0.634(48)	−0.123(62)	−0.004(59)	0.817(24)
Measured $\|+Y\rangle$	−0.028(58)	−0.650(45)	0.005(61)	0.825(23)
Measured $\|+Z\rangle$	−0.081(65)	−0.083(66)	0.849(31)	0.924(16)
Measured $\|-X\rangle$	−0.645(43)	0.135(59)	0.030(63)	0.823(22)
Measured $\|-Y\rangle$	0.026(65)	0.719(40)	−0.013(61)	0.860(20)
Measured $\|-Z\rangle$	0.032(58)	−0.069(58)	−0.736(39)	0.868(19)

Without any corrections (tomography errors or wrong charge state), we obtain the following prepared states, with average fidelity F=0.807(10):

Client	Server			
	⟨X⟩	⟨Y⟩	⟨Z⟩	Fidelity
Measured $\|x\rangle$	0.534(55)	−0.090(62)	0.009(62)	0.767(27)
Measured $\|y\rangle$	0.024(60)	−0.582(51)	−0.013(62)	0.791(26)
Measured $\|0\rangle$	−0.073(69)	−0.072(69)	0.786(42)	0.893(21)
Measured $\|-x\rangle$	−0.552(49)	0.143(61)	0.055(63)	0.776(24)
Measured $\|-y\rangle$	0.052(64)	0.623(47)	−0.018(62)	0.811(23)
Measured $\|1\rangle$	0.030(57)	−0.028(55)	−0.606(46)	0.803(23)

When only applying tomography error correction, we find an average preparation fidelity F=0.829(9):

Client	Server			
	⟨X⟩	⟨Y⟩	⟨Z⟩	Fidelity
Measured $\|x\rangle$	0.573(49)	−0.096(59)	0.010(58)	0.786(24)
Measured $\|y\rangle$	0.025(56)	−0.624(45)	−0.014(59)	0.812(23)
Measured $\|0\rangle$	−0.078(64)	−0.077(65)	0.843(32)	0.921(16)
Measured $\|-x\rangle$	−0.592(44)	0.153(57)	0.059(59)	0.796(22)
Measured $\|-y\rangle$	0.056(61)	0.667(41)	−0.020(59)	0.834(20)
Measured $\|1\rangle$	0.032(54)	−0.030(53)	−0.650(40)	0.825(20)

4.6 Discussion

In summary, we have demonstrated the operation of a link layer and a physical layer for entanglement-based quantum networks. The link layer abstracts the entanglement generation procedure provided by the physical layer — implemented here with two NV center-based quantum network nodes — into a robust platform-independent service that can be used to run quantum networking applications. We performed full quantum state tomography of the states delivered by the link layer, tested its ability to deliver states at different fidelities in real-time, and verified remote state preparation of a qubit from the client on the server, a fundamental step towards blind quantum computation [7]. We have shown that our implementation of link and physical layers can deliver entangled states at the fidelity requested by the user, despite some marginal inefficiencies — some of which can be addressed in a future version of the protocols (e.g. avoiding Pauli corrections unless necessary). We have also quantified the additional latency incurred by deploying the link layer protocol on top of the physical layer. Although not detrimental, the extra overhead is still noticeable, but can also be scaled down by optimizing the scheduling of entanglement generation requests. We also acknowledge that scheduling a quantum node's resources is still an open problem [35, 41, 42] and that the simple approach taken here is likely a suboptimal choice for more advanced quantum networks. We emphasize, however, that our link layer protocol is not tied to any particular scheduling algorithm or architecture — it merely expects that the schedule of each node be matched with its peer. In Ref. [11] for example, the schedule was instead formed via a distributed queue protocol, and in the future other architectures and algorithms [35] may be more suitable for scaling to larger networks.

Other research challenges posed by our work include an in-depth analysis of the security of quantum network implementations. For example, it is clear that if the classical control messages used in our protocol are not authenticated, unwanted entanglement generation may be triggered at one of the nodes. In some physical layer implementations such as the one considered here, this may negatively impact the quality of the qubits already stored at the node [20], and hence impact availability. Initial work indicates that the performance impact of adding authentication, however, is small (refer to Chapter 6).

The adoption of the techniques presented here (which are not specific to our diamond devices) by other quantum network platforms [27, 32, 33, 36, 37, 38, 39] will boost the development towards large-scale and heterogeneous quantum networks. Real-time control of memory qubits, as well as the availability of multi-node networks and dynamic network schedules, will enable demonstrations of the higher layers of the network stack [24], which in turn will open the door to end-to-end connectivity on a platform-independent quantum network.

References

[1] M. H. Abobeih, J. Cramer, M. A. Bakker, N. Kalb, M. Markham, D. J. Twitchen, and T. H. Taminiau. "One-second Coherence for a Single Electron Spin Coupled to a Multi-qubit Nuclear-spin Environment". In: *Nature Commun.* 9.1 (2018), pp. 1–8. DOI: 10.1038/s41467-018-04916-z.

[2] A. Aguado, V. López, J. P. Brito, A. Pastor, D. R. López, and V. Martin. "Enabling Quantum Key Distribution Networks via Software-Defined Networking". In: *ONDM*. IEEE, 2020, pp. 1–5. DOI: 10.23919/ONDM48393.2020.9133024.

[3] M. Alshowkan, B. P. Williams, P. G. Evans, N. S. Rao, E. M. Simmerman, H.-H. Lu, N. B. Lingaraju, A. M. Weiner, C. E. Marvinney, Y.-Y. Pai, B. J. Lawrie, N. A. Peters, and J. M. Lukens. "Reconfigurable Quantum Local Area Network Over Deployed Fiber". In: *PRX Quantum* 2 (4 2021), p. 040304. URL: 10.1103/PRXQuantum.2.040304.

[4] L. Aparicio, R. Van Meter, and H. Esaki. "Protocol Design for Quantum Repeater Networks". In: *AINTEC*. ACM, 2011, pp. 73–80. DOI: 10.1145/2089016.2089029.

[5] 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". In: *Nature* 497 (2013), pp. 86–90. DOI: 10.1038/nature12016.

[6] 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". In: *Phys. Rev. X* 9.3 (2019), pp. 031045-1–031045-12. DOI: 10.1103/PhysRevX.9.031045.

[7] A. Broadbent, J. Fitzsimons, and E. Kashefi. "Universal Blind Quantum Computation". In: *FOCS*. IEEE, 2009, pp. 517–526. DOI: 10.1109/FOCS.2009.36.

[8] M. Caleffi. "Optimal Routing for Quantum Networks". In: *IEEE Access* 5 (2017), pp. 22299–22312. DOI: 10.1109/ACCESS.2017.2763325.

[9] K. Chakraborty, D. Elkouss, B. Rijsman, and S. Wehner. "Entanglement Distribution in a Quantum Network: A Multicommodity Flow-Based Approach". In: *IEEE Transactions on Quantum Engineering* 1 (2020), pp. 1–21. DOI: 10.1109/TQE.2020.3028172.

[10] K. Chakraborty, F. Rozpędek, A. Dahlberg, and S. Wehner. "Distributed Routing in a Quantum Internet". 2019. arXiv: 1907.11630.

[11] A. Dahlberg, M. Skrzypczyk, T. Coopmans, L. Wubben, F. Rozpędek, M. Pompili, A. Stolk, P. Pawełczak, R. Knegjens, J. de Oliveira Filho, R. Hanson, and S. Wehner. "A Link Layer Protocol for Quantum Networks". In: *SIGCOMM*. ACM, 2019, pp. 159–173. DOI: 10.1145/3341302.3342070.

[12] A. Dahlberg, B. van der Vecht, C. Delle Donne, M. Skrzypczyk, I. te Raa, W. Kozlowski, and S. Wehner. "NetQASM—A Low-Level Instruction Set Architecture for Hybrid Quantum–Classical Programs in a Quantum Internet". In: *Quantum Science and Technology* 7.3 (2022), p. 035023. DOI: 10.1088/2058-9565/ac753f.

[13] A. Delteil, Z. Sun, W.-b. Gao, E. Togan, S. Faelt, and A. Imamoğlu. "Generation of Heralded Entanglement Between Distant Hole Spins". In: *Nat. Phys.* 12.3 (2016), pp. 218–223. DOI: 10.1038/nphys3605.

[14] A. D. Ferguson et al. "Orion: Google's Software-Defined Networking Control Plane". In: *NSDI*. USENIX Association, 2021, pp. 83–98. URL: https://www.usenix.org/conference/nsdi21/presentation/ferguson.

[15] C. Granade, J. Combes, and D. G. Cory. "Practical Bayesian tomography". In: *New Journal of Physics* 18.3 (2016), p. 033024. DOI: 10.1088/1367-2630/18/3/033024.

[16] C. Granade, C. Ferrie, I. Hincks, S. Casagrande, T. Alexander, J. Gross, M. Kononenko, and Y. Sanders. "QInfer: Statistical Inference Software for Quantum Applications". In: *Quantum* 1 (2017), p. 5. DOI: 10.22331/q-2017-04-25-5.

[17] L. Gyongyosi and S. Imre. "Decentralized Base-Graph Routing for the Quantum Internet". In: *Phys. Rev. A* 98.2 (2018), p. 022310. URL: 10.1103/PhysRevA.98.022310.

[18] J. Hofmann, M. Krug, N. Ortegel, L. Gérard, M. Weber, W. Rosenfeld, and H. Weinfurter. "Heralded Entanglement Between Widely Separated Atoms". In: *Science* 337.6090 (2012), pp. 72–75. DOI: 10.1126/science.1221856.

[19] 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”. In: *Nature* 558.7709 (2018), pp. 268–273. DOI: 10.1038/s41586-018-0200-5.

[20] N. Kalb, P. C. Humphreys, J. J. Slim, and R. Hanson. “Dephasing Mechanisms of Diamond-Based Nuclear-Spin Memories for Quantum Networks”. In: *Phys. Rev. A* 97.6 (2018), p. 062330. DOI: 10.1103/PhysRevA.97.062330.

[21] N. Kalb, A. A. Reiserer, P. C. Humphreys, J. J. W. Bakermans, S. J. Kamerling, N. H. Nickerson, S. C. Benjamin, D. J. Twitchen, M. Markham, and R. Hanson. “Entanglement Distillation Between Solid-State Quantum Network Nodes”. In: *Science* 356.6341 (2017), pp. 928–932. DOI: 10.1126/science.aan0070.

[22] W. Kozlowski, A. Dahlberg, and S. Wehner. “Designing a Quantum Network Protocol”. In: *CoNEXT*. ACM, 2020, pp. 1–16. DOI: 10.1145/3386367.3431293.

[23] W. Kozlowski, F. Kuipers, and S. Wehner. “A P4 Data Plane for the Quantum Internet”. In: *EuroP4*. ACM, 2020, pp. 49–51. URL: 10.1145/3426744.3431321.

[24] W. Kozlowski and S. Wehner. “Towards Large-Scale Quantum Networks”. In: *NANOCOM*. ACM, 2019, pp. 1–7. DOI: 10.1145/3345312.3345497.

[25] D. L. Moehring, P. Maunz, S. Olmschenk, K. C. Younge, D. N. Matsukevich, L.-M. Duan, and C. Monroe. “Entanglement of Single-Atom Quantum Bits at a Distance”. In: *Nature* 449 (2007), pp. 68–71. DOI: 10.1038/nature06118.

[26] B. Nachman, M. Urbanek, W. A. de Jong, and C. W. Bauer. “Unfolding Quantum Computer Readout Noise”. In: *npj Quantum Information* 6.1 (2020), pp. 1–7. DOI: 10.1038/s41534-020-00309-7.

[27] 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 ”. In: *Phys. Rev. Lett.* 123.18 (2019), p. 183602. URL: 10.1103/PhysRevLett.123.183602.

[28] M. Pant, H. Krovi, D. Towsley, L. Tassiulas, L. Jiang, P. Basu, D. Englund, and S. Guha. “Routing Entanglement in the Quantum Internet”. In: *npj Quantum Information* 5.1 (2019), pp. 1–9. DOI: 10.1038/s41534-019-0139-x.

[29] A. Pirker and W. Dür. “A Quantum Network Stack and Protocols for Reliable Entanglement-Based Networks”. In: *New Journal of Physics* 21.3 (2019), p. 033003. URL: 10.1088/1367-2630/ab05f7.

[30] 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”. In: *Science* 372.6539 (2021), pp. 259–264. DOI: 10.1126/science.abg1919.

[32] S. Ritter, C. Nölleke, C. Hahn, A. Reiserer, A. Neuzner, M. Uphoff, M. Mücke, E. Figueroa, J. Bochmann, and G. Rempe. "An Elementary Quantum Network of Single Atoms in Optical Cavities". In: *Nature* 484.7393 (2012), pp. 195–200. DOI: 10.1038/nature11023.

[33] B. C. Rose, D. Huang, Z.-H. Zhang, P. Stevenson, A. M. Tyryshkin, S. Sangtawesin, S. Srinivasan, L. Loudin, M. L. Markham, A. M. Edmonds, D. J. Twitchen, S. A. Lyon, and N. P. de Leon. "Observation of an Environmentally Insensitive Solid-State Spin Defect in Diamond". In: *Science* 361.6397 (2018), pp. 60–63. DOI: 10.1126/science.aao0290.

[34] S. Shi and C. Qian. "Concurrent Entanglement Routing for Quantum Networks: Model and Designs". In: *SIGCOMM*. ACM, 2020, pp. 62–75. DOI: 10.1145/3387514.3405853.

[35] M. Skrzypczyk and S. Wehner. "An Architecture for Meeting Quality-of-Service Requirements in Multi-User Quantum Networks". 2021. arXiv: 2111.13124.

[36] N. T. Son, C. P. Anderson, A. Bourassa, K. C. Miao, C. Babin, M. Widmann, M. Niethammer, J. Ul Hassan, N. Morioka, I. G. Ivanov, F. Kaiser, J. Wrachtrup, and D. D. Awschalom. "Developing Silicon Carbide for Quantum Spintronics". In: *Appl. Phys. Lett.* 116.19 (2020), p. 190501. DOI: 10.1063/5.0004454.

[37] L. J. Stephenson, D. P. Nadlinger, B. C. Nichol, S. An, P. Drmota, T. G. Ballance, K. Thirumalai, J. F. Goodwin, D. M. Lucas, and C. J. Ballance. "High-Rate, High-Fidelity Entanglement of Qubits Across an Elementary Quantum Network". In: *Phys. Rev. Lett.* 124.11 (2020), p. 110501. DOI: 10.1103/PhysRevLett.124.110501.

[38] R. Stockill, M. J. Stanley, L. Huthmacher, E. Clarke, M. Hugues, A. J. Miller, C. Matthiesen, C. Le Gall, and M. Atatüre. "Phase-Tuned Entangled State Generation between Distant Spin Qubits". In: *Phys. Rev. Lett.* 119.1 (2017), p. 010503. DOI: 10.1103/PhysRevLett.119.010503.

[39] M. E. Trusheim, B. Pingault, N. H. Wan, M. Gündoğan, L. De Santis, R. Debroux, D. Gangloff, C. Purser, K. C. Chen, M. Walsh, J. J. Rose, J. N. Becker, B. Lienhard, E. Bersin, I. Paradeisanos, G. Wang, D. Lyzwa, A. R.-P. Montblanch, G. Malladi, H. Bakhru, A. C. Ferrari, I. A. Walmsley, M. Atatüre, and D. Englund. "Transform-Limited Photons From a Coherent Tin-Vacancy Spin in Diamond". In: *Phys. Rev. Lett.* 124.2 (2020), p. 023602. DOI: 10.1103/PhysRevLett.124.023602.

[40] R. Van Meter, T. Satoh, T. D. Ladd, W. J. Munro, and K. Nemoto. "Path Selection for Quantum Repeater Networks". In: *Networking Science* 3.1 (2013), pp. 82–95. DOI: 10.1007/s13119-013-0026-2.

[41] G. Vardoyan, S. Guha, P. Nain, and D. Towsley. "On the Capacity Region of Bipartite and Tripartite Entanglement Switching". In: *SIGMETRICS Perform. Eval. Rev.* 48.3 (2021), pp. 45–50. DOI: 10.1145/3453953.3453963.

[42] G. Vardoyan, S. Guha, P. Nain, and D. Towsley. "On the Stochastic Analysis of a Quantum Entanglement Switch". In: *Perform. Eval. Rev.* 47.2 (2019), pp. 27–29. DOI: 10.1145/3374888.3374899.

[43] M. Pompili, C. Delle Donne, I. te Raa, B. van der Vecht, M. Skrzypczyk, G. M. Ferreira, L. de Kluijver, A. J. Stolk, S. L. N. Hermans, P. Pawełczak, W. Kozlowski, R. Hanson, and S. Wehner. *Data and software supporting "Experimental Demonstration of Entanglement Delivery Using a Quantum Network Stack"*. 2021. DOI: `10.4121/16912522`.

[44] *FreeRTOS Real-Time Operating System for Microcontrollers.* Amazon Web Services. URL: `https://www.freertos.org/` (visited on Feb. 28, 2023).

[45] *NetQASM SDK*. QuTech. URL: `https://github.com/QuTech-Delft/netqasm` (visited on Feb. 28, 2023).

[46] *The White Rabbit Project*. CERN. URL: `https://white-rabbit.web.cern.ch/` (visited on Feb. 28, 2023).

5

Quantum Networking With an Elementary Operating System

An operating system (OS) for quantum network nodes should provide more than just networking functionalities. Ultimately, it should enable quantum networking applications to be written in high-level, platform-independent software, and should be able to manage the resources of the underlying device when deployed in a multi-node and multi-user quantum network. This chapter discusses our implementation of QNodeOS, an OS for quantum network nodes, which includes a quantum network stack for entanglement generation, as well as resource management and scheduling features that allow the concurrent execution of multiple applications. We also design and propose a set of benchmarks which will be used to quantify — in upcoming work — the performance of the OS on state-of-the-art quantum network hardware based on nitrogen-vacancy centers in diamond.

This chapter is extracted from the article in preparation: **C. Delle Donne**, M. Iuliano, B. van der Vecht, M. Skrzypczyk, I. te Raa, G. M. Ferreira, T. van der Steenhoven, A. R.-P. Montblanch, M. Pompili, S. L. N. Hermans, N. Demetriou, B. van Ommen, T. H. Taminiau, P. Pawełczak, W. Kozlowski, R. Hanson, and S. Wehner. "QNodeOS: An Operating System for Quantum Network Nodes".

Contributions: as indicated in Chapter 2.

The preliminary experiments conducted in Chapter 4 showcased elementary quantum networking functionalities through a platform-independent control system — mainly, the quantum networking stack embedded in QNodeOS. Nevertheless, entanglement generation is just one of the blocks constituting fully-fledged quantum communications applications, which also include local quantum processing and classical communication and processing, as shown in Figure 2.2. With QNodeOS, we aim to take the state of the art of quantum networking experiments one step further, and demonstrate the execution of complete applications, some of which comprise quantum and classical processing and communication. We also include one case study that serves as a proof-of-concept demonstration of the usefulness of a multitasking-ready OS, to be expanded on when investigating multi-user quantum networks more in depth. In this chapter we describe the proposed test applications, which will serve as case studies for the upcoming evaluation of QNodeOS on NV center devices. Prior to that, we also give an overview of the implementation of QNodeOS and the underlying QDevice. Refer to Appendix A for additional details on the implementation of the components of QNodeOS and their interfaces, and to Appendix B for the specification of the interface to the QDevice.

5.1 Implementation

Figure 5.1 outlines software and hardware implementation of QNodeOS and the whole node system. QNodeOS is implemented in C++ on top of FreeRTOS [4], a tiny operating system for microcontrollers. The stack runs on a dedicated MicroZed [5] — an off-the-shelf platform based on the Zynq-7000 SoC, which hosts two ARM Cortex-A9 processing cores, of which only one is used, clocked at 667 MHz. QNodeOS connects to peer QNodeOS systems via TCP over a Gigabit Ethernet interface. We opted for a device like the Zynq-7000 SoC for its advantageous trade-off between high flexibility and moderate cost (around € 100 in the Netherlands at the time of writing). Whilst more concrete device requirements may arise from our future benchmarking of QNodeOS, we believe that the selected SoC provides enough computational bandwidth for the envisioned tasks, and it also offers the possibility to implement optimized hardware modules on its FPGA fabric. We however remark that the design of QNodeOS is in no way tied to a certain computing architecture. For the QDevice, we replicated the setup used for Chapter 4, which mainly consists of: (1) an ADwin-Pro II [3] acting as the main orchestrator of the setup; (2) a

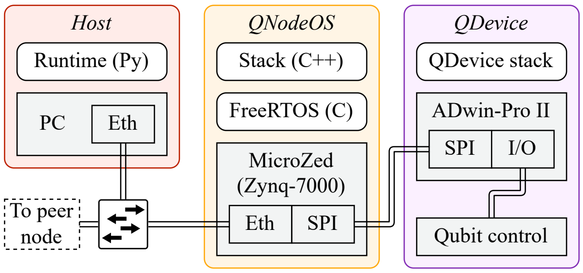


Figure 5.1: Node deployment overview. Our quantum network node consists of a desktop machine for the host runtime, a Zynq-7000 SoC for QNodeOS, and a series of digital and analog controllers for the QDevice.

			Lines of code			
Component	**File type**	**Files**	**Total**	**Blanks**	**Comments**	**Code**
Core code	C/C++	85	16273	2779	1953	11541
	C/C++ header	121	13281	2418	4188	6675
	CMake	25	486	99	43	344
	Assembly	1	141	15	0	126
Test code	C/C++	57	21849	3842	2195	15812
	C/C++ header	17	1725	288	177	1260
	Python	14	2577	527	427	1623
	CMake	25	483	97	22	364
Estimated schedule effort (COCOMO)			15.47 months			
Estimated people required (COCOMO)			11.81			

Table 5.1: Code metrics for QNodeOS core code and testing code, including number of files per language, lines of code, and *constructive cost model* (COCOMO) effort estimates [1], generated using scc [6]. The COCOMO metrics were generated using the "Semi-detached" model, meaning that the estimated are computed assuming that the project requires a certain level of expertise and creativity and the problem is not well understood (i.e. there is research involved).

series of subordinate devices responsible for qubit control, including laser pulse generators and optical readout circuits; (3) the quantum physical device, based on NV centers, counting one single (communication) qubit for each node. The QDevice is where the time-critical qubit control lies. QNodeOS interfaces with the QDevice's ADwin-Pro II through a 12.5 MHz SPI interface, used to exchange 4-byte control messages at a rate of 50 kHz. Finally, the host layer is a Python runtime running on a general-purpose 4-core desktop machine running Linux. The host machine connects to QNodeOS via TCP over the same Gigabit Ethernet interface that QNodeOS uses to connect to its peers (average ping RTT of 0.1 ms), and sends application registration requests and quantum code blocks over this interface (10 to 1000 bytes, depending on the length of the block).

QNodeOS is a complex project, developed by multiple researchers and engineers over the course of around three years and counting. Its test infrastructure is also relatively large, with a continuous-integration pipeline consisting of an extensive set of unit tests for each of the OS core components and some system-level application tests. Table 5.1 reports code metrics for QNodeOS code code and testing code, including number of files per language, lines of code, and *constructive cost model* (COCOMO) effort estimates [1], generated using scc [6]. Although these metrics do not fully capture the research effort put into QNodeOS, they are an indicator of the amount of engineering work involved. We also point out that QNodeOS builds on top of existing real-time software frameworks — namely, FreeRTOS. We implemented QNodeOS on top of FreeRTOS to avoid re-implementing standard OS primitives like threads and network communication. FreeRTOS provides basic OS abstractions like tasks, inter-task message passing, and the TCP/IP stack. The FreeRTOS kernel — like any other standard OS — cannot however directly manage the quantum resources (qubits, entanglement requests and entangled pairs), and hence its task scheduler cannot take decisions based on such resources. QNodeOS adds these capabilities and takes care of the scheduling of quantum code blocks based on the status of the quantum resources.

5.2 Test Cases

We propose a set of benchmarks that are to be used to verify the functioning of QNodeOS. These four case studies are aimed at validating (1) single-node execution, including qubit initialization, gates, and measurements, (2) entanglement generation, (3) delegated quantum computation, and (4) multitasking.

5.2.1 Single-Qubit Gate Tomography

Our first case study is a simple local application where a single gate is applied to a qubit initialized in the $|0\rangle$ state, and then the qubit is measured in a number of bases. This translates to one or more single-qubit gates and one qubit measurement. The application is run several times to assess the quality of the prepared state, and various qubit states are analyzed. This single-qubit gate tomography is the simplest application QNodeOS can run — there is a single user process running, and the network process is not even activated, given that entanglement is never requested.

Configurations and expected results. This application is configured to apply six different gates in separate runs: $R_x(\pi)$, $R_x(\pi/2)$, $R_x(\pi/4)$, $R_y(\pi)$, $R_y(\pi/2)$, $R_y(\pi/4)$. Each resulting state is measured in all six cardinal bases. This application is run 1000 times for each combination of gate and readout basis. We expect the measured state fidelity to be in line with what the quantum hardware is capable of delivering, demonstrating that the overhead incurred by QNodeOS is negligible, at least when running local applications.

5.2.2 Entanglement Generation

The second test case is an application that generates an entangled pair between two nodes and then measures the generated state. This is a distributed application, where both nodes are active — they engage in entanglement generation, and they both measure their end of the entangled pair. As the user can specify the requested fidelity of the entangled pairs, this application is to be run for various target fidelities. This time, all QNodeOS components are at work. Since entanglement is requested, the quantum network stack is triggered, and thus the network process becomes active, competing for resources with the user process. The QMMU is also invoked by the network process to transfer ownership of the entangled qubit to the user process (the inter-process communication primitive of QNodeOS).

Configurations and expected results. Entanglement generation is run for a range or target fidelities: 0.50, 0.55, 0.60, 0.65, 0.70, 0.75 and 0.80. Entangled pairs are read out in various bases to measure their correlators $\langle XX\rangle$, $\langle YY\rangle$ and $\langle ZZ\rangle$ (and their $\pm$ variations, for a total of 12 correlators). The application is run 125 times for each combination of target fidelity and correlator, for a total of 10 500 entangled pairs. We expect the measured fidelity to be at least matching — within measurement uncertainty — the requested minimum fidelity, if the quantum hardware is capable of delivering such requested fidelities.

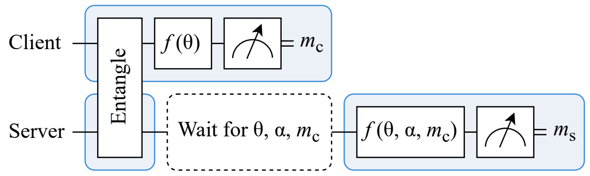


Figure 5.2: Schematic of delegated computation application. The client wishes to have the server perform a quantum computation on a certain data qubit. To do so, the two nodes establish entanglement, then the client processes and measures its end of the entangled pair, sends the computation parameter to the server, which finally executes the delegated computation. Blue boxes represent quantum code blocks. The client's application is composed of a single block, while the server's consists of one block for entanglement and one block for the quantum computation, interleaved by a classical code block (the reception of the computation parameter).

5.2.3 Delegated Computation

With this case study we aim to showcase a more complex quantum network application, schematically depicted in Figure 5.2. Here, one node acts as the client, and the other as the server. The client's goal is to delegate a certain quantum computation on some data qubit to the server, while keeping the server agnostic to the computation. To perform the desired computation, described by a parameter α, the two nodes follow these steps: (1) the two node establish an entangled pair, (2) the client "encodes" its qubit by means of a series of local gates, described by a parameter θ, (3) the client measures its end of the entangled pair and stores the classical outcome m_c (4) the client communicates the delegated computation parameter, which is a function of α, θ and m_c, (5) the server performs the computation, (6) the server measures its end of the entangled pair and sends the classical outcome m_s back to the client. In this scheme, the client application consists of a single quantum code block and an additional classical code block that communicates the computation parameter to the server. More interestingly, the server application comprises two quantum code blocks — the first is the establishment of the entangled pair, and the second is the delegated computation — interleaved by a classical code block that receives the computation parameter from the client. This is the first example of an application with inter-block and inter-node data dependencies, where the execution (on the server) spans more than one quantum code block, and thus the quantum state generated in one block has to persist and remain valid for the other block too.

Configurations and expected results. The delegated computation is run for various values of α (π, $\pi/2$) and θ (π, $\pi/2$, $\pi/4$). The application is run 500 times for each combination of α and θ. The metrics of interest are: (1) the measured fidelity of the qubit state after the delegated computation for each of the computation values, (2) a breakdown of the average application latency, to give an indication of where time is spent during execution. We expect the fidelity to be somewhat lower than the best-case performance due to the communication latency incurred at the application level — the "Wait for θ, α, m_c" block in Figure 5.2. In the latency breakdown, we also expect this classical communication step at the application level to be the dominating factor. These delays are expected to manifest in distributed applications. The resulting idle times can be allocated to other pending applications.

5.2.4 Multitasking

One of the core features of modern OSes is the ability to run several applications concurrently, a key aspect in multi-user nodes and networks. QNodeOS is designed with multitasking capabilities — not only can it multiplex a user process and the network process, but it also allows for multiple user processes to run at the same time. This means that multiple users can submit their applications simultaneously, and QNodeOS will service all pending user processes based on resource availability, in order to increase the utilization of the QDevice and to limit idle time and average application latency. One possible shortcoming of multitasking on a quantum network node is the trade-off between concurrency and fidelity: applications that have active data in the quantum memory, and that are waiting to be scheduled while other applications are in progress, may experience lower-quality qubit states, given that such quality degrades due to the passing of time and to the noise induced by operations on other qubits.

We aim to demonstrate the multitasking capabilities of QNodeOS by having multiple users run independent applications concurrently. In our case study, a pool of users runs the delegated computation application, while the rest of the users runs the local single-qubit gate tomography application. Multitasking is evaluated on the client node, while the server just runs its part of the delegated computation application. The idle time resulting from running the delegated computation application on the client is a perfect candidate for scheduling other pending applications. The multitasking performance of QNodeOS is assessed under various system load conditions, which essentially depend on the number of users submitting applications QNodeOS at the same time. We note that, even though a higher degree of concurrency should in principle results in better device utilization, this is limited by the scarce physical resources available on the underlying QDevice.

Configurations and expected results. Device utilization and average application latency are measured on the client, for various configurations of users and applications: N users, with $N \in \{2, 3, 5, 10\}$, half (rounded up) of which running the local single-gate tomography application, and the remaining half (rounded down) running the delegated computation application. To measure the performance benefit of multitasking, we also run the same set of applications with multitasking disabled — on QNodeOS, this means that a user process can only be scheduled if no other user processes are either running or waiting for entanglement generation. We expect the fidelity of the states measured at the end of both applications to be somewhat comparable across the configurations with and without multitasking enabled, given that the underlying QDevice has a one-qubit memory at the moment, and thus the level of concurrency is fairly limited. On the other hand, we expect device utilization and average application latency to be affected by the multitasking capabilities of QNodeOS. Whilst this is a relatively simple benchmark and the degree of concurrency is highly limited by the available quantum memory, this case study should exemplify why it is important for an OS for quantum network nodes to be multitasking-capable, and how such an OS can take advantage of idle times.

5.3 Discussion

The test cases discussed in this chapter complement those of Chapter 4, and represent a milestone for the evaluation of operating systems for quantum network nodes. These experiments are planned for QNodeOS for the near future. The results of this experimental effort will primarily showcase certain aspects of the importance of software abstractions for quantum networking. Additionally, the outcome will provide insights into design and implementation strengths an pitfalls on QNodeOS, and establish a baseline performance for similar studies involving future versions of our system or alternative designs.

In the next and final chapter of this thesis we analyze the impact of data origin authentication of classical messages exchanged at the quantum network stack level. The results of this investigation will serve as guidelines for incorporating data origin authentication mechanisms into more advanced designs of QNodeOS.

References

[1] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece. *Software Cost Estimation with COCOMO II*. Prentice Hall Press, 2009.

[3] *ADwin-Pro II*. Jäger GmbH. URL: `https://www.adwin.de/us/produkte/proII.html` (visited on Feb. 28, 2023).

[4] *FreeRTOS Real-Time Operating System for Microcontrollers*. Amazon Web Services. URL: `https://www.freertos.org/` (visited on Feb. 28, 2023).

[5] *MicroZed Development Board*. Avnet. URL: `https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/microzed/` (visited on Feb. 28, 2023).

[6] `scc`*: Code Counter With Complexity Calculations and COCOMO Estimates*. Ben Boyter. URL: `https://github.com/boyter/scc` (visited on Feb. 28, 2023).

6 Data Origin Authentication in the Quantum Networking Stack

Quantum networking protocols make use of classical communications to coordinate entanglement generation and other tasks. In this chapter, we discuss the need for authentication of classical messages exchanged at the quantum network stack level, with focus on concrete protocol proposals. We then experimentally measure the overhead incurred by sending authenticated classical messages through an authentication system that uses key material supplied by an MDI-QKD system. We use this information to simulate the performance of a quantum link whose protocol stack uses an authenticated classical channel, and compare that to existing simulations of the same protocol stack. We find that message authentication overhead is not detrimental to entanglement generation and to the fidelity of the entangled pairs.

This chapter is based on the article in preparation: J. S. Abrahams, **C. Delle Donne**, P. Brussee, T. Middelburg, R. C. Berrevoets, J. A. Slater, and S. Wehner. "Data Origin Authentication in the Quantum Network Protocol Stack".

Contributions: J. S. Abrahams, **C. Delle Donne**, J. A. Slater and S. Wehner designed the research. J. S. Abrahams researched and implemented MACs. P. Brussee, T. Middelburg and R. C. Berrevoets prepared the authentication proxy. J. S. Abrahams and **C. Delle Donne** prepared the framework to measure RTT delays. J. S. Abrahams and **C. Delle Donne** ran the simulations. J. S. Abrahams, **C. Delle Donne**, J. A. Slater and S. Wehner wrote the manuscript with input from all authors. J. A. Slater and S. Wehner supervised the research. J. S. Abrahams and **C. Delle Donne** contributed equally to this work.

Up until now, several proposals have been put forward as to how quantum networks should be structured [9, 16, 17]. Researchers are also investigating how to abstract the complex physics of quantum network devices so as to provide platform-independent services to the end user [7, 8, 13] — one very basic service would be the distribution of entanglement between network nodes, that is, *entanglement generation*. To coordinate this, and other activities, proposals for a *quantum network stack* have provided outlines of the layers and the separation of responsibilities, as well as protocols to populate these layers [7, 10]. Within the proposed stack, each layer makes use of the service exposed by the layer below, and provides a higher-level service to the layer above [7]. This stack is inspired by classical architectures like the well-known TCP/IP network stack or the more generic Open System Interconnect (OSI) model, and is illustrated in Figure 6.1. Each layer and protocol within the quantum network stack makes use of classical control messages, exchanged over a classical network, to coordinate the quantum communication activities. These messages from the *control protocols* form what we refer to as the *classical data plane* — which lives alongside the *quantum data plane*, in which information encoded in quantum systems is transmitted.

When designing control protocols for quantum networks, one should carefully estimate various key performance indicators, such as their impact on the end-to-end latency of quantum services, such as entanglement generation between network nodes. One obvious reason to assess the impact on latency, is the fundamental aspect of quantum information which happens to impose strict constraints on end-to-end latency: *decoherence*. Storing quantum data reliably for extended periods of time is non-trivial. Qubits have relatively short lifetimes, usually of the order of milliseconds, or at best of a few seconds [1, 5]. Therefore, not only can high end-to-end latency affect the quality of the service offered by the network, but in some cases it may result in no service at all. Classical processing and communication overhead must, thus, be kept to a minimum, such that the generated entangled qubits can be used as quickly as possible.

On top of that, control messages must be transmitted in a secure manner for a quantum network to function reliably. Satoh et al. [15] list forged classical messages as a general concern for quantum networks. To prevent such forgeries, quantum network nodes may employ data origin authentication (DOA) cryptography suites, to distinguish between gen-

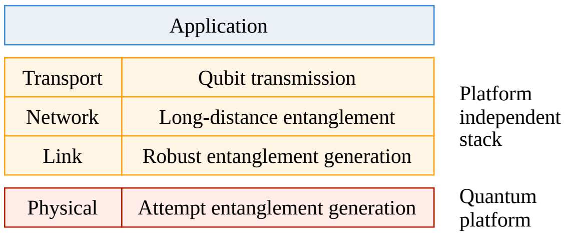

Application
Transport
Qubit transmission
Network
Long-distance entanglement
Link
Robust entanglement generation
Platform independent stack
Physical
Attempt entanglement generation
Quantum platform

Figure 6.1: Functional allocation of layers in a quantum network stack, adapted from Ref. [7]. The physical layer is quantum platform-dependent. The link layer provides platform-independent robust entanglement generation. All subsequent layers are therefore also platform independent, including the network and optionally transport layers which facilitate end-to-end entanglement between non-adjacent nodes. The application layer uses the services offered by the stack to perform quantum networking tasks.

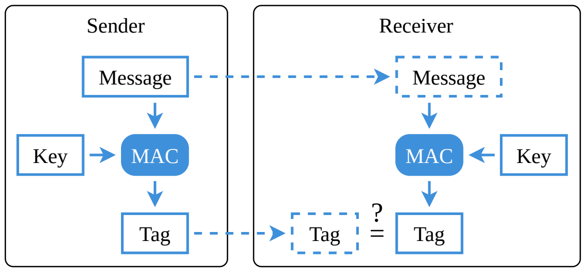


Figure 6.2: General structure of a message authentication code (MAC), where sender and receiver have a shared key. The receiver checks the output of the MAC to verify that the message was not modified in transit.

uine and fraudulent control messages. DOA is performed using a secret that is shared between two parties. Then, a message authentication code (MAC) algorithm can produce a tag for each control message both at the sending and at the receiving end, to verify that the message contents (1) were not altered and (2) were produced by a party which owns the shared secret. This process — known as *authentication* — is illustrated in Figure 6.2.

Inevitably, performing DOA on control messages would incur some computation and communication overhead. Such latency is typically neglected when modeling, simulating, or experimentally validating network protocols for quantum communications. In this work, we examine the effects of latency caused by classical DOA on a simulated quantum network link requested to generate entanglement between two nodes (henceforth referred to as the simulated quantum link). First, we experimentally measure the latency incurred by a DOA system using QKD-powered authentication algorithms. We then analyze the behavior of the simulated quantum link — using a simulator for quantum networks [6] — where the model of the control channel of the network includes the measured latencies incurred by the QKD-powered DOA. The contributions of this work are as follows:

1. We provide a concise motivation for why DOA is a necessary component to uphold the availability of system-level protocols of the quantum network stack, and to help maintain the integrity of quantum data.
2. We experimentally measure the added latencies incurred by DOA, performed using a QKD-based authentication system, when run on a classical communication link.
3. We offer a quantitative analysis of the impact of DOA, using the latencies measured as per the previous point, when applied to the quantum network protocols of a simulated quantum link (such link was first studied by Dahlberg et al. [7]).

6.1 Related Work

Conventional networking technology is an essential component of quantum networks and quantum networking applications. Kozlowski and Wehner mention that the security of classical communications is of concern when designing a quantum network [11]. Satoh et al. present a general motivation for authenticating the control channel of a quantum link [15]. They model attack vectors on quantum communications through the lens of confidentiality, integrity, and availability. Without any security measures in place, an attacker may:

- Disrupt the network in any number of ways, affecting its *availability*.
- Interfere with data sent via the network, hampering the *integrity* of quantum data.
- Read out quantum data through the accompanying classical control data, affecting the *confidentiality* of quantum data.

All identified proposals of quantum network designs and quantum network protocols use some form of conventional communication to control and coordinate quantum communication [7, 9, 10, 11, 13, 16, 17]. In this work, we investigate one such proposal for a quantum network stack: the one put forth by Dahlberg et al. [7], which has been evaluated in simulation, as well as on hardware [14], and extended by Kozlowski et al. [10].

The proposed protocol stack for quantum networks includes physical, link, network, transport, and application layers, as illustrated in Figure 6.1. The physical layer protocol, called midpoint heralding protocol (MHP) [7], performs heralded entanglement generation attempts. At the link layer, the quantum entanglement generation protocol (QEGP) [7] protocol has an internal retry mechanism and performs coordination between adjacent nodes to provide more robust entanglement generation. QEGP accepts two types of requests from the layer above: (1) create and keep (CK), to create an entangled pair and store it in memory; (2) measure directly (MD), to create entangled pair, but measure it immediately, and report its outcome. At the network layer, the quantum network protocol (QNP) [10] protocol coordinates entanglement generation and swap operations on a chain of nodes between two non-adjacent end nodes.

In this work, we simulate the three service-level protocols MHP, QEGP, and QNP. In the next section, we explain why data origin authentication (DOA) is important for these protocols to function, mostly focusing on *availability* of the network and *integrity* of quantum data.

6.2 Why Data Origin Authentication

We investigate the applicability of data origin authentication to three system-level protocols for quantum network stacks: MHP, QEGP and QNP [7, 10]. Quantum applications and their protocols themselves lie outside the scope of this work. Here, we provide a non-exhaustive example list of actions that a malicious actor may perform if they were able to forge or modify classical messages exchanged at the control protocol level. We mention whether each action affects the *availability* of the link or network, or the *integrity* of the quantum data sent via the network.

Physical layer. MHP operates at the physical layer. Hardware vulnerabilities of the physical entanglement generation process are outside the scope of this work.

Example 6.2.1. *Availability.* Change a successful heralding signal to an error code such Alice and Bob falsely conclude that entanglement has failed.

Example 6.2.2. *Integrity.* Modify a signal from a heralding station by changing the state announcement such that Alice or Bob apply the wrong Pauli corrections to their qubits.

Example 6.2.3. *Integrity.* Interfere with mismatch verification [7, 14] such that Alice and Bob falsely conclude that a MHP request belongs to the same QEGP request, thus hampering the integrity of quantum data due to cross-process interference.

Link layer. QEGP operates at the link layer [7]. As proposed in Ref. [7], it is used to synchronize entanglement requests, and to communicate the number of available memory qubits and the expiration of requests.

Example 6.2.4. *Availability.* Continually send requests for entanglement, exhausting the resources of nodes receiving them [11].

Example 6.2.5. *Availability.* When advertising the number of available communication qubits or storage qubits, set either to `0`. The receiving node then assumes that there are no communication or storage qubits available on the sending node.

Example 6.2.6. *Integrity.* Change the qubit identifier of an entanglement generation request such that Alice and Bob entangle the wrong data qubits, causing cross-path or process interference.

Network layer. QNP operates at the network layer. Conceptually, it allows non-adjacent nodes in a quantum network to coordinate entanglement generation. This is akin to the internet protocol (IP) in the classical stack. It makes use of `FORWARD` and `TRACK` messages to track entanglement generation [10, Figure 6]. `FORWARD` messages are used to communicate entanglement requests, and `TRACK` messages contain classical correction information for both end-nodes in the network.

Example 6.2.7. *Availability.* Modify `FORWARD` message so that intermediary nodes do not receive entanglement swap instructions.

Example 6.2.8. *Integrity.* Modify `TRACK` messages such that the end-nodes of a path do not receive the correct entanglement swap outcome, and thus apply the wrong corrections to their qubits.

We illustrate the concerns of availability and data integrity in a quantum network stack in Figure 6.3. Such security threats are addressed in part by using data origin authentication, which can prevent modification and forgery of classical control messages. It should be noted, however, that availability may still be hampered by an adversary capable of halting transmission of classical control messages outright. Furthermore, if a malicious actor were to gain access to a node itself then this might circumvent many or all security mechanisms in place, including DOA, and should therefore also be prevented.

In general, we also note that classical control messages might reveal information about who is using the network and the types of operations being performed. Therefore, in a more mature network, it will likely be worthwhile to also encrypt the contents of control messages. Satoh et al. [15] mention tracking as a general concern for inter-node classical communication within the quantum stack. Encryption combined with transmission of random noise could address tracking concerns.

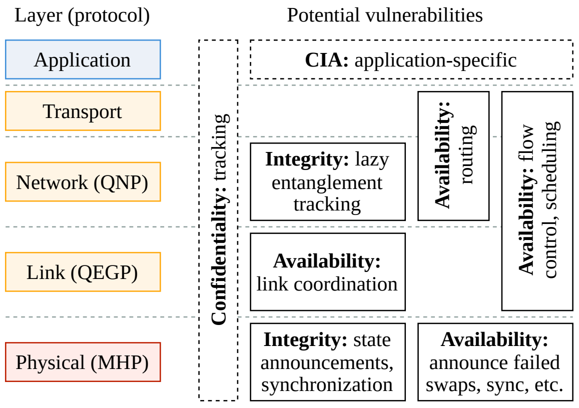

Layer (protocol)
Potential vulnerabilities
Application
Transport
Network (QNP)
Link (QEGP)
Physical (MHP)
Confidentiality: tracking
CIA: application-specific
Integrity: lazy entanglement tracking
Availability: routing
Availability: flow control, scheduling
Availability: link coordination
Integrity: state announcements, synchronization
Availability: announce failed swaps, sync, etc.

Figure 6.3: A summary of example effects of tampering with control messages at each layer of the quantum network stack through the lens of confidentiality, integrity, and availability (CIA). We take as examples concrete implementations of quantum network protocols — including QNP, QEGP and MHP [7, 10] — to highlight the potential vulnerabilities at each layer of the stack. In this work, we do not focus on confidentiality issues, as identified by Satoh et al. [15].

6.3 Experimental Methodology

We aim to examine the effects of data origin authentication, and the total latency incurred by the classical control messages, on the performance of a hypothetical quantum link. To limit the scope of our investigation, we focus on the performance of link and physical layer protocols as proposed by Dahlberg et al. [7], and we extend the simulation therein such that it fully captures (1) classical transmission overhead, roughly defined as the latency of classical messages through the network and all networking equipment, and (2) DOA overhead, roughly defined as the processing time required by the DOA systems.

Instead of modeling transmission and authentication overhead analytically, we measure it experimentally over a real-world, physically deployed network, consisting of multiple datacenter locations, sample quantum node controllers running FreeRTOS [18] on a MicroZed board [19], and complete DOA servers. The DOA servers exist at both ends of our real-world network, and run message authentication protocols. From a networking perspective, these servers act like transparent proxy servers, and are thus generally unknown and unseen to the quantum controllers, but do authenticate control messages between the two ends of our real-world deployed network. The DOA servers authenticate (and validate) control messages using key material obtained from a physically-deployed QKD system, also running between the datacenter locations. The QKD devices form a measurement-device independent quantum key distribution (MDI-QKD) network, based on the work by Berrevoets et al. [4]. The QKD devices run next to the DOA servers, and deliver QKD-key material via standard interface protocols.

We recorded round-trip time (RTT) statistics of sample control messages sent between the quantum controllers — and thus through the message authentication protocols of the QKD-powered DOA servers. We then injected these RTT data, along with other simulation parameters described below, to extract metrics, as done in Ref. [7], and assess the

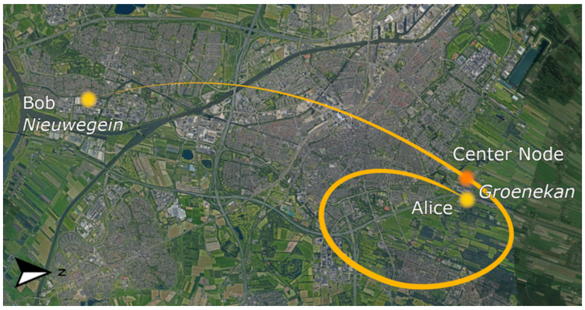


Figure 6.4: Physical layout of Alice and Bob nodes. Alice is located in Groenekan, the Netherlands, while Bob is in Nieuwegein, the Netherlands. The optical fiber connecting Alice to the center node is 20 km in length, whereas the connection between Bob and the center node is 22 km.

performance of a simulated quantum entanglement generating link The measured control communication overheads is reported in Section 6.4, while the simulation results are presented in Section 6.5.

DOA servers. We collect control communication latency measurements under three different configurations of the DOA servers:

1. Bypass servers: at first, we bypass the DOA servers altogether. This is useful to measure baseline communication latency, excluding all computation delays that would be introduced by the servers.
2. No MAC: this time, the DOA servers are configured to skip the authentication (and verification) step, but packets do go through the servers' processing pipelines. Results for this configuration give us insights into the overhead of processing packets on the DOA servers, excluding the computation latency incurred by the MAC authentication (and verification) step.
3. Poly1305-AES: finally, we configure the DOA servers to use Poly1305-AES [3], which is a popular, fast, and computationally secure MAC. In this configuration, a DOA server attaches a 128-bit mac to each outgoing message.

We do not report on a configuration where the DOA servers use an information-theoretically secure MAC, as such an authentication scheme would consume considerably more key material than the QKD system could deliver. We did verify this assessment by configuring the DOA servers to use VMAC [12] — an information-theoretically secure MAC — during which the DOA servers attempted to use considerably more key material than available.

Network topology. In our network deployment, we name the two end locations Alice and Bob. As is normal in MDI-QKD, each end location is connected to a center hub. In the real-world network, the link connecting Alice to the center hub runs for 20 km, whereas the link between Bob and the center hub is 22 km in length. Thus, the RTT we measure is for a link of a total length of 42 km. The topology of Alice, Bob, and the center hub of the MDI-QKD system is illustrated in Figure 6.4. The control communication overhead

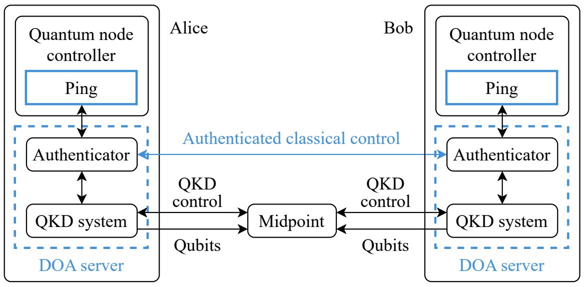


Figure 6.5: Experimental setup used to measure the end-to-end delays of transmitting an *authenticated classical message* from Alice to Bob with 42 km of fiber-optic cables between them. Classical messages go through a DOA server that tags messages using MDI-QKD-generated key material.

measured in Section 6.4 will be scaled to simulate quantum links of various lengths (2.5 km to 75 km) in Section 6.5.

6.4 Classical Communication Latency

In order to obtain an estimate of the expected latency incurred by classical control messages in a quantum network stack, we measure the round-trip time (RTT) of a sample of messages when sent through an authenticated classical channel in the field. The messages are ICMP (ping) packets, sized to mimic MHP and QEGP packets. The authentication mechanism is external to the controllers exchanging ping messages, and runs on transparent DOA proxy servers between them. The DOA server next to the sender calculates a MAC tag over the sent message, forwards the message and the tag to the receiver, whose DOA servers verifies the tag before delivering the message to the destination controller. The experimental setup is depicted in Figure 6.5.

Ping messages are exchanged between two real-time classical network nodes similar to those used in the experimental validation of QEGP [14] — MicroZed boards [19] running a FreeRTOS [18] application — connected to the DOA servers via Gigabit Ethernet interfaces. The sending node records the RTT of the message, and computes various statistics on these timestamps, most notably average and standard deviation. We use these statistics to extrapolate the expected control communication latency for the various quantum link simulation scenarios presented in Section 6.5.

The message authentication code (MAC) on both ends retrieves authentication keys from a local QKD node running a QKD key server, which contains key material identical to its counterpart at the other end. Key material is produced by an MDI-QKD implementation deployed in the Utrecht area, in the Netherlands. The MDI-QKD system is very similar to the one implemented in Ref. [4].

Rate and size of messages. When using our particular DOA servers, one cannot transmit a classical message more than once every 10 ms without experiencing detrimental packet loss. The reason for this is that the DOA servers run prototype software designed for research purposes, and have not been optimized for performance. We thus transmit ping messages at a rate of 100 Hz. Moreover, the employed DOA servers can only authenticate packets with a payload that is small enough to be accommodated (together with all

Server	**Payload**	**RTT** [µs]	
configuration	[Bytes]	**mean**	**std**
Bypass servers	12	3657	23
	1200	3881	22
No MAC	12	14 036	797
	1200	14 089	767
Poly1305-AES	12	14 099	861
	1200	14 015	729

Table 6.1: Mean and standard deviation of round-trip time (RTT) for different configurations of the DOA servers and for different message sizes. Measurements for the "Bypass servers" configuration follow a simple Gaussian distribution. Measurements for the other two configurations can be approximated by bimodal Gaussian distributions, with one of the two modes strongly dominating over the other. This table reports mean and standard deviation of the strongest modes of each distribution. In the "Poly1305-AES" configuration, the DOA servers attaches a 128-bit key to each message.

protocol headers and the MAC tag) in a single Ethernet frame. Therefore, we send ping messages with payloads of (1) 12 bytes, which is the size of the smallest QEGP message, and (2) 1200 bytes, which is close to the maximum allowed by the DOA server. The second configuration is useful to examine how much RTTs depend on the size of the message.

Results. We record the round-trip time of 360,000 ping messages per configuration (DOA server configuration and size of message). Mean and standard deviation of the measured RTTs are reported in Table 6.1, whilst the raw distributions of RTT measurements are presented in Appendix D.

When bypassing the DOA servers altogether, RTT delays follow a simple Gaussian distribution, with a mean of less than 4 ms and a standard deviation of around 20 µs. When messages go through the DOA servers (configurations "No MAC" and "Poly1305-AES"), RTT measurements can be approximated by bimodal Gaussian distributions (see Appendix D). This bimodal distribution is likely the result of some caching behavior exhibited by the DOA servers. For both configurations, one of the two modes strongly dominates over the other, with the mean of this mode (around 14 ms for all configurations and sizes) being approximately equal to the overall mean of the entire distribution (±2 % difference at most). Table 6.1 reports mean and standard deviation of the strongest modes of each distribution, which are deemed to be the most representative statistics to use in our simulations. Refer to Appendix D for a more in-depth analysis of the RTT distributions.

The noticeable gap between "Bypass servers" and the other two configurations is an indicator of the suboptimal packet processing performance of the DOA server, which is merely a soft-processing packet pipeline implemented in Python. The computational overhead introduced by the actual MAC is overshadowed by the baseline latency of the DOA servers, as observed in the "Poly1305-AES" configuration. Interestingly, the size of messages does not appear to be a noticeable factor in the mean end-to-end latency.

We can therefore conclude that latency is dominated by two main factors: (1) the performance of any classical networking hardware and the length of the classical link, which depend on the network technology and topology, and (2) the packet processing performance of the DOA servers. However, in a real production network, it is fair to assume that packets will be processed at a much faster rate, and thus result in an end-to-end latency that is much more similar to that of the "Bypass servers" configuration. Considering also that authentication does not incur any noticeable overhead — as shown by the difference between the "No MAC" and "Poly1305-AES" configurations — these results support the case for both the use of DOA in quantum networks, as well as the use of DOA supported by QKD in present-day, real-world production networks.

6.5 Simulation Results

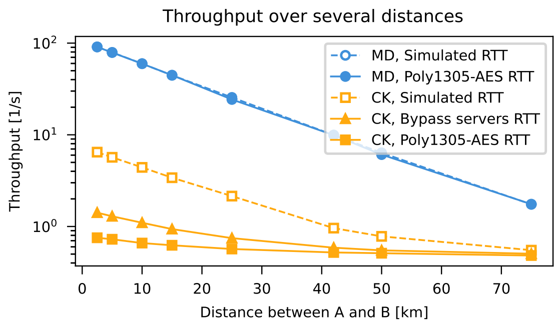
We quantify the effects of using an authenticated classical channel for control messages exchanged at the quantum network stack level. In particular, we augment the model used by Dahlberg et al. [7] to simulate the performance of physical (MHP) and link (QEGP) layer control protocols within the quantum network stack. As opposed to the original work, we model the classical control communication latency to also include transmission and authentication overhead as experimentally measured in our real-world network, and presented in Section 6.4. We report the *throughput* of the quantum link, expressed as number of entangled pairs generated per second.

We use RTT statistics from Table 6.1 for our simulations. As described in Section 6.4, RTT measurements for the "Bypass servers" configuration can be approximated by a Gaussian distribution. RTTs for the "No MAC" and "Poly1305-AES" configurations instead follow bimodal Gaussian distributions, with one of the modes dominating over the other. For these two configurations, we use mean and standard deviation of the strongest mode.

Configurations. We run our simulations for two types of configurations: (1) To begin with, we analyze the quantum link throughput for several node-to-node distances (2.5 km to 75 km), at a fixed requested fidelity ($F_{\text{min}} = 0.65$). For the various distances, we scale the classical delays measured in Section 6.4 accordingly. For this configuration, we compare our augmented model with the original baseline, in which transmission and authentication delays were not modeled [7]. (2) Furthermore, we analyze the quantum link throughput at a fixed node-to-node distance (42 km between the two nodes, same as in the real-world network in Section 6.4), but this time varying the requested fidelity at the QEGP layer (fidelity 0.50 to 0.85). This time, we compare three models, corresponding to the three configurations of the DOA servers as per Section 6.4: "Bypass servers", "No MAC", and "Poly1305-AES".

For each of the above configurations, we perform 20 simulation runs, each consisting of 20 minutes of continuous use of the quantum link. We then calculate the average throughput over all runs for each configuration with a confidence interval of 95 %.

Results. The results for throughput versus distance are illustrated in Figure 6.6. For measure directly (MD) type requests (for which entanglement can be measured directly, as described in earlier), mean throughput is approximately equal across the two models of added latency. This is to be expected, given that these types of requests can be effectively pipelined — that is, the next entanglement request can be initiated right after the previous one without the need to wait for any acknowledgment messages from a heralding station — and thus throughput is mostly dominated by the physical entanglement gen-



	Breakdown of distance A – B [km]							
A – M	1.5	3	6	9	15	22	30	45
M – B	1	2	4	6	10	20	20	30
Total	2.5	5	10	15	25	42	50	75

Figure 6.6: Throughput (rate) of entangled pair generation for multiple distances between node A and B, with a single midpoint station in between and for a requested fidelity of $F_{\text{min}} = 0.65$. Classical communication delays were modeled using latency measurements collected as described in Section 6.4 — configurations "Bypass servers" and "Poly1305-AES" — as well as replicated from the original simulations of QEGP and MHP [7]. The simulated RTT configuration represents the best-case scenario, given that only propagation delays are modeled in this one. The configuration "Poly1305-AES" is the worst-case scenario instead, given that it models delays measured on the field, including the overhead incurred by the slow packet processing pipeline. For MD type requests, only the best case and the worst case are plotted, since they practically overlap, and thus any other scenario in between best and worst would not result in any significant difference. The table below the plot shows how distance is distributed between A, B, and the midpoint station M. The 25 km data point is equivalent to the QL2020 hypothetical setup simulated in Ref. [7].

eration procedure, and not as much by classical communication latency. On the other hand, create and keep (CK) requests (for which entanglement must be stored while waiting for acknowledgment messages, as described earlier) show a variation in throughput that depends on the delay model. The best-case scenario — corresponding to the original simulations of MHP and QEGP [7] ("Simulated RTT") — results in the highest throughput, which peaks at around 6.47 pairs per second at the shortest distance. This is followed by the scenario with "Bypass servers" delays, where classical communication latency has a higher overhead than the previous case, but is more realistic. In this case, the shortest link can deliver around 1.42 entangled pairs per second. Finally, the worst-case scenario of "Poly1305-AES", where classical communication delays also include the overhead of the slow packet processing pipeline of the DOA servers, can yield a little less than one pair per second. This decrease in throughput is expected, but fortunately, it is also not detrimental to the point that the quantum link becomes non-functional altogether. Naturally, final numbers depend strongly on the quantum properties of the quantum memories (quantum coherence time). Conceivably, if the coherence time of the quantum memories was longer than the added latency of DOA servers, the decrease in performance would be less significant.

The results for throughput versus fidelity are shown in Figure 6.7. Again, the outcome

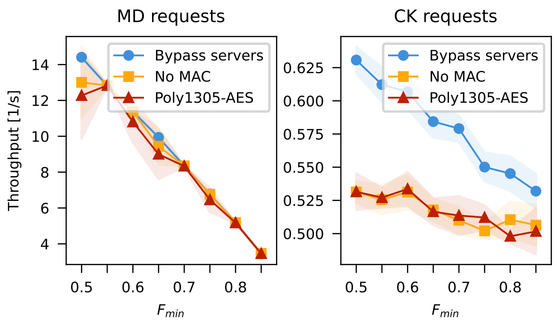


Figure 6.7: Throughput (rate) of entangled pair generation for multiple requested fidelities, with 20 km between node A and the midpoint station, and 22 km between the midpoint station and node B, and for three configurations of the proxy as per Section 6.4: Bypass servers, No MAC, and Poly1305-AES. The solid line represents mean throughput, the colored area around it depicts the 95 % confidence interval. Measure directly (MD) type requests are not as heavily affected as these are pipelined. Create and keep (CK) type requests are performed sequentially, and thus increases in classical delays have a more profound impact on throughput.

matches our expectations. MD requests are not significantly affected by classical communication, and their throughput only decreases when higher-fidelity entangled pairs are requested. CK requests, instead, are more affected by classical communication latency, and their throughput is a lot lower than that of MD requests. Additionally, throughput also decreases when classical messages go through the DOA servers (“Bypass servers” versus “No MAC”). As expected from the results in Table 6.1, the extra latency incurred by the actual MAC computation are negligible, and its effect on throughput not noticeable (“No MAC” versus “Poly1305-AES”).

To summarize, our experiments and simulations demonstrate that (1) in real-world quantum links, throughput is likely to be worse than what results from simulations that do not fully model transmission delays of messages stemming from latencies of conventional communication networks, (2) the overhead incurred by classical communications is sizeable, but does not outright disrupt the operation of quantum links, and (3) delays incurred by data origin authentication are negligible.

6.6 Discussion

We have shown how the classical data plane of a quantum network stack presents a significant attack surface for confidentiality, integrity, and availability of the quantum link and data therein. To address these concerns one must employ, among other things, data origin authentication on the classical control messages exchanged at the quantum network stack level. We conclude that data origin authentication is necessary to both uphold the integrity of quantum data and the availability of the quantum network itself.

We have also simulated the performance of a hypothetical quantum link under the assumption that control messages are authenticated by conventional DOA techniques. Here, we modeled classical communication latency, including authentication overhead and transmission delay, using metrics collected from a real-world communication link, authenticated using DOA servers, using key material sourced from a QKD system. If we disregard the large packet processing delays incurred by the DOA servers itself — the delays of the servers packet pipeline, and not the delays of the authentication software running on the server — we observe that an authenticated classical control channel introduces a negligible amount of extra classical overhead.

However, we have also seen that even just transmission delays have a noticeable effect on the performance of a quantum link, whether or not data origin authentication is applied — entanglement generation throughput drops when classical communication delays are larger. Nevertheless, we have observed in our simulations that even when the entangled qubits must be stored in quantum memories with limited lifetimes, the quantum link remains operational.

References

[1] M. H. Abobeih, J. Cramer, M. A. Bakker, N. Kalb, M. Markham, D. J. Twitchen, and T. H. Taminiau. "One-second Coherence for a Single Electron Spin Coupled to a Multi-qubit Nuclear-spin Environment". In: *Nature Commun.* 9.1 (2018), pp. 1–8. DOI: 10.1038/s41467-018-04916-z.

[3] D. J. Bernstein. "The Poly1305-AES Message-Authentication Code". In: *FSE*. Springer, 2005, pp. 32–49. DOI: 10.1007/11502760_3.

[4] R. C. Berrevoets, T. Middelburg, R. F. 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". In: *Communications Physics* 5.1 (2022), pp. 1–8. DOI: 10.1038/s42005-022-00964-6.

[5] 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". In: *Phys. Rev. X* 9.3 (2019), pp. 031045-1–031045-12. DOI: 10.1103/PhysRevX.9.031045.

[6] T. Coopmans, R. Knegjens, A. Dahlberg, D. Maier, L. Nijsten, J. Oliveira, M. Papendrecht, J. Rabbie, F. Rozpędek, M. Skrzypczyk, L. Wubben, W. de Jong, D. Podareanu, A. Torres-Knoop, D. Elkouss, and S. Wehner. "NetSquid, a NETwork Simulator for QUantum Information using Discrete events". In: *Communications Physics* 4.1 (2021), p. 164. DOI: 10.1038/s42005-021-00647-8.

[7] A. Dahlberg, M. Skrzypczyk, T. Coopmans, L. Wubben, F. Rozpędek, M. Pompili, A. Stolk, P. Pawełczak, R. Knegjens, J. de Oliveira Filho, R. Hanson, and S. Wehner. "A Link Layer Protocol for Quantum Networks". In: *SIGCOMM*. ACM, 2019, pp. 159–173. DOI: 10.1145/3341302.3342070.

[8] J. Illiano, M. Caleffi, A. Manzalini, and A. S. Cacciapuoti. "Quantum Internet Protocol Stack: a Comprehensive Survey". 2022. arXiv: 2202.10894.

[9] S. K. Joshi, D. Aktas, S. Wengerowsky, M. Lončarić, S. P. Neumann, B. Liu, T. Scheidl, G. C. Lorenzo, Ž. Samec, L. Kling, A. Qiu, M. Razavi, M. Stipčević, J. G. Rarity, and R. Ursin. "A Trusted Node–Free Eight-User Metropolitan Quantum Communication Network". In: *Science Advances* 6.36 (2020), eaba0959. DOI: 10.1126/sciadv.aba0959.

[10] W. Kozlowski, A. Dahlberg, and S. Wehner. “Designing a Quantum Network Protocol”. In: *CoNEXT*. ACM, 2020, pp. 1–16. DOI: 10.1145/3386367.3431293.

[11] W. Kozlowski and S. Wehner. “Towards Large-Scale Quantum Networks”. In: *NANOCOM*. ACM, 2019, pp. 1–7. DOI: 10.1145/3345312.3345497.

[12] T. Krovetz. “Message Authentication on 64-Bit Architectures”. In: *Selected Areas in Cryptography*. Springer, 2007, pp. 327–341. DOI: 10.1007/978-3-540-74462-7_23.

[13] A. Pirker and W. Dür. “A Quantum Network Stack and Protocols for Reliable Entanglement-Based Networks”. In: *New Journal of Physics* 21.3 (2019), p. 033003. URL: 10.1088/1367-2630/ab05f7.

[14] M. Pompili, C. Delle Donne, I. te Raa, B. van der Vecht, M. Skrzypczyk, G. M. 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”. In: *npj Quantum Information* 8.1 (2022), p. 121. DOI: 10.1038/s41534-022-00631-2.

[15] T. Satoh, S. Nagayama, S. Suzuki, T. Matsuo, and R. Van Meter. “Attacking the Quantum Internet”. In: *IEEE Transactions on Quantum Engineering* 2 (2021), pp. 1–17. DOI: 10.1109/TQE.2021.3094983.

[16] E. Schoute, L. Mancinska, T. Islam, I. Kerenidis, and S. Wehner. “Shortcuts to Quantum Network Routing”. 2016. arXiv: 1610.05238.

[17] R. Van Meter and J. Touch. “Designing Quantum Repeater Networks”. In: *IEEE Communications Magazine* 51.8 (2013), pp. 64–71. DOI: 10.1109/MCOM.2013.6576340.

[18] *FreeRTOS Real-Time Operating System for Microcontrollers*. Amazon Web Services. URL: https://www.freertos.org/ (visited on Feb. 28, 2023).

[19] *MicroZed Development Board*. Avnet. URL: https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/microzed/ (visited on Feb. 28, 2023).

7 Conclusion

Most of the results obtained in this thesis were aimed at laying the foundations for a quantum internet where nodes are programmable, and can be interacted with by anyone with basic knowledge of quantum networking, as opposed to only being operable by quantum physics experts. Whilst this is merely a single step towards more scalable quantum networks, we have demonstrated the importance of abstractions in this emerging field, and we hope this work can be of inspiration to future research seeking to continue on this path.

7.1 Summary of Results

The primary focus of this work was to design, build and experimentally validate *software abstractions* for *programmable* quantum network nodes. Here we recap the main results of our investigation.

Design considerations. To begin with, we have explored the fundamental reasons that make quantum networking challenging, and why classical networking and operating system (OS) research fall short of such challenges. We have thus laid out general design considerations to be addressed when thinking about abstractions for quantum network nodes. With respect to that, we have also put forward a shortlist of the main constituent parts of an OS for such nodes — which include a network stack for quantum communications, a process scheduler, and a quantum memory management unit. These design considerations were presented in Chapter 3.

Design of an operating system for quantum network nodes. Following our design considerations, we have produced a first design of an OS for quantum network nodes. Our system, which we call QNodeOS, includes all the components as per the design considerations, and represents a fully-functional proof of concept to be used for further research on the topic. Key design points of QNodeOS are: (1) the integration of a state-of-the-art quantum networking stack, to coordinate entanglement generation in a platform-independent manner; (2) the separation of quantum networking and local operations into distinct processes, to allow for better scheduling decisions and higher concurrency; (3) the isolation of quantum platform-specific abstractions into a device driver, called QDevice driver, to enable deploying QNodeOS onto various quantum platform with minimal effort. Our design was presented in Chapter 3.

Entanglement generation. As a first case study of QNodeOS, we tested its quantum networking capabilities by means of three simple applications centered around entanglement generation. We have implemented the quantum networking stack proposed in recent publications [1, 2], integrated it into QNodeOS, and demonstrated its operation on a two-node state-of-the-art quantum network based on NV center technology. With these preliminary tests, we showed how OS abstractions can simplify the programming of quantum network nodes at a minimal, almost negligible, performance cost. We in fact found that, while the quantum networking stack introduces latency as compared to "bare-metal" entanglement generation at the physical layer, the application-level fidelity of the delivered states is not much affected by this overhead. These results were presented in Chapter 4.

Quantum networking applications and multitasking. To further showcase the service offered by QNodeOS and to establish a performance baseline for an OS for quantum network nodes, we proposed a set of fully-fledged quantum networking applications, which will be tested on QNodeOS as part of upcoming work. These test cases consist of delegated quantum computation protocol, as well as the concurrent execution of multiple applications, which will leverage the multitasking capabilities of QNodeOS. Once obtained, the results of this evaluation will assert the importance of a robust stack of abstraction layers for quantum networking. We presented these test cases in Chapter 5.

Data origin authentication in quantum networking. In perspective of a large-scale, fully-operational quantum network, where classical communications ought to be more secure than in a laboratory, we have also analyzed the performance penalty that would be incurred if classical messages exchanged at the quantum networking stack were to be authenticated so as to ensure integrity. Using a combination of experimentally-measured authentication delays and the simulation of a quantum link, we have shown that the delay incurred by data origin authentication would not be detrimental to the fidelity of the delivered entangled states. This topic was investigated in Chapter 6.

7.2 Future Work

We hope that our work and results open the door to further research on the topic of OSes for quantum networking. On that regard, we have identified a few areas of this field that would benefit from a deeper investigation.

Further experimental validation of QNodeOS. The natural next step for the evaluation of QNodeOS is to run the test cases presented in Chapter 5 to push the boundaries of experimental demonstrations of quantum networking even further. As discussed already, this experimental validation will serve both to construct a baseline performance overview of OSes for quantum networking and to demonstrate the capabilities of such systems.

Quantum network node architecture. In our design, we have defined a certain architecture of a quantum network node, and we have not investigated possible alternatives. In particular, the separation between host, QNodeOS and QDevice determines which component is responsible for which task, and limits certain scheduling decisions to certain components. In the future, one might want to explore other solutions and examine scenarios in which, for instance, resource allocation and scheduling decisions are not only taken on QNodeOS.

Scheduling of processes and applications. The rudimentary process scheduler embedded in QNodeOS is just good enough to guarantee concurrency and to prioritize what in general is deemed to have higher priority in quantum networking applications — that is, entanglement generation. Nevertheless, an optimal scheduler might need to factor in more run-time data to take better scheduling decisions that are context-dependent. For instance, one could well imagine a scheduler which is aware of the approximate duration of the tasks to be scheduled, as well as of the current level of decoherence of the quantum states in memory, and thus re-compute task priorities at run-time when needed. Additionally, as the responsibilities of QNodeOS and host might be redefined (see previous paragraph), one could design the host such that it takes part in making scheduling decisions, or at least supporting QNodeOS with application- and device-specific hints.

Integration with control plane. At the moment, the interactions of QNodeOS with the control plane of a quantum network are mostly one-sided — the control plane installs network schedules on QNodeOS, mostly. In a future iteration of QNodeOS, and in a full-blown implementation of a control plane, the former could feed a set of run-time statistics back into the control plane scheduler, so as to provide an up-to-date picture to the latter, which can in turn refine the computing of the network schedule in real-time.

Smart quantum memory management unit. The quantum memory management unit (QMMU) implemented on the first version of QNodeOS is rather basic. As discussed in Chapter 3, a more advanced iteration of the QMMU might feature smart mechanisms for tracking (estimating) qubit decoherence to offer better run-time scheduling support. Additionally, it could have access to device-specific descriptions of the underlying quantum memory to improve allocation strategies that better cater to memory lifetime requirements and to the limitations of near-term quantum devices.

Larger-scale experimental validation and benchmarking. As quantum networks become larger in scale, and the nodes therein begin to offer a larger set of quantum resources, a natural experimental follow-up of this thesis would see QNodeOS, or any future OS for quantum networks, be tested again more complex quantum networking applications, especially those requiring more qubits, and those involving more parties within a network. Along the same lines, a great addition to this work would be a set of experimental tests where QNodeOS is deployed on a larger variety of quantum physical platforms, eventually even on a heterogeneous network of several different devices.

References

[1] A. Dahlberg, M. Skrzypczyk, T. Coopmans, L. Wubben, F. Rozpędek, M. Pompili, A. Stolk, P. Pawełczak, R. Knegjens, J. de Oliveira Filho, R. Hanson, and S. Wehner. “A Link Layer Protocol for Quantum Networks”. In: *SIGCOMM*. ACM, 2019, pp. 159–173. DOI: 10.1145/3341302.3342070.

[2] W. Kozlowski, A. Dahlberg, and S. Wehner. “Designing a Quantum Network Protocol”. In: *CoNEXT*. ACM, 2020, pp. 1–16. DOI: 10.1145/3386367.3431293.

A
QNodeOS Components and Interfaces

We provide additional details on the components of the QNodeOS architecture and their interfaces. Figure A.1 gives an overview of all the components of QNodeOS. The *process manager* marshals accesses to all user and kernel processes. The *scheduler* assigns ready processes to the *processor*, which runs quantum instructions through the underlying QDevice, processes classical QASM instructions locally, and registers entanglement requests with the *entanglement management unit* (EMU). The EMU maintains a list of EPR sockets and entanglement requests, forwards the latter to the *quantum network stack*, which, in turn, registers available entangled qubits with the EMU. Finally, the *quantum memory management unit* (QMMU) keeps track of used qubits, and transfers qubit ownership across processes when requested.

Process manager. The process manager owns QNodeOS processes and marshals accesses to those. Creating a process, adding a routine to it and accessing the process's data must be done through the process manager. Additionally, the process manager is used by other components to notify *events* that occur inside QNodeOS, upon which the state of one of more processes is updated. Process state updates result in a notification to the scheduler.

The process manager exposes interfaces for three services:

- *Process management* (interface ① in Figure A.1): to create and remove processes, and to add routines to them. When the user registers an application, the QNodeOS API Handler uses the process manager to create a QNodeOS user process. The returned process ID can be later used to add a routine to that process, or to remove the process once all its routines are fully processed.
- *Event notification* (interface ② in Figure A.1): to notify that an event has occurred inside QNodeOS, including the addition of a routine, the completion of a routine, the scheduling of the process, the hitting of a *wait* condition, and the generation of an entangled qubit destined to the process. Some events trigger follow-up actions — for instance, when a process that was waiting for an event becomes ready, it gets added to the queue of ready processes maintained by the scheduler.

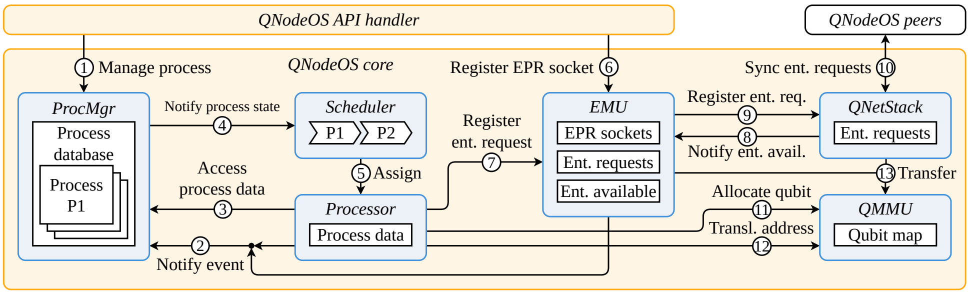


Figure A.1: QNodeOS core components and internal interfaces. The core layer includes: (1) a *process manager* (ProcMgr), which owns and manages access to QNodeOS processes; (2) a *scheduler*, responsible for selecting the next process to be run; (3) a *processor*, which processes routines' instructions; (4) an *entanglement management unit* (EMU), which keeps a list of entanglement requests and available entangled qubits; (5) a *quantum network stack* (QNetStack), whose responsibility is to coordinate with peer nodes to schedule quantum networking instructions; (6) a *quantum memory management unit* (QMMU), which keeps a record of allocated qubits.

- *Process data access* (interface ③ in Figure A.1): to access a process's routines and its classical memory space, mostly used while running the process (through the processor).

Scheduler. The QNodeOS scheduler registers processes that are ready to be scheduled, and assigns them to the QNodeOS processor when the latter is available. Ready processes are stored in a *prioritized ready queue*, and processes of the same priority are scheduled with a first-come-first-served policy.

The scheduler only exposes one interface for process state notifications (interface ④ in Figure A.1), used by the process manager to signal when a process transitions to a new state. When a QNodeOS process transitions to the ready state, it is directly added to the scheduler's prioritized ready queue. When a process becomes idle, or is waiting for an event to happen, the scheduler simply registers that the processor has become available.

Processor. The QNodeOS processor handles the execution of QNodeOS user and kernel processes, by running classical instructions locally and issuing quantum instructions to the QDevice driver. While executing a process, the processor reads its routines and accesses (reads and writes) its classical memory. The processor implements a specific instruction set architecture dictated by the QASM language of choice.

The processor exposes one interface for processor assignment (interface ⑤ in Figure A.1), used by the QNodeOS scheduler to activate the processor, when it is idling, and assign it to a QNodeOS process.

Entanglement management unit. The entanglement management unit (EMU) maintains a list of open *EPR sockets* and a list of *entanglement requests*, and keeps track of the *entangled qubits* produced by the quantum network stack. Received entanglement requests are considered valid only if an EPR socket associated to such requests exists. Valid requests are forwarded to the quantum network stack. Entangled qubit generations are notified as events to the process manager.

The EMU exposes interfaces for three services:

- *EPR socket registration* (interface ⑥ in Figure A.1): to register and open EPR sockets belonging to an application, and to set up internal classical network tables and to establish classical network connection.
- *Entanglement request registration* (interface ⑦ in Figure A.1): to add entanglement requests to the list of existing ones, to be used when matching produced entangled qubits with a process that requested them.
- *Entanglement notification* (interface ⑧ in Figure A.1): to register the availability of an entangled qubit, produced by the quantum network stack, and to link it to an existing entanglement request.

Quantum network stack. The quantum network stack on QNodeOS follows the model presented in Ref. [1] which is based on the classical OSI network stack model for separation of responsibilities. In particular, *data link layer* and *network layer* protocols are part of the quantum network stack on QNodeOS. The *physical layer* is implemented on the QDevice, the *application layer* is part of the Host, and all remaining layers are not currently part of the stack.

The quantum network stack component has an associated *QNodeOS kernel process*, created statically on QNodeOS. However, this process's routine is dynamic: the instructions to be executed on the processor depend on the outstanding entanglement generation requests received from EMU and network peers.

The quantum network stack exposes interfaces for two services:

- *Entanglement request registration* (interface ⑨ in Figure A.1): to add entanglement requests coming from the EMU to the list of existing ones, which are used to fill in the quantum network stack process's routine with the correct instructions to execute.
- *Entanglement request synchronization* (interface ⑩ in Figure A.1): similar to the entanglement request registration interface, but to be used to synchronize (send and receive) requests with QNodeOS network peers.

Quantum memory management unit. The quantum memory management unit (QMMU) receives requests for *qubit allocations* from QNodeOS processes, and manages the subsequent usage of those. It also translates QASM *virtual qubit addresses* into physical addresses for the QDevice, and keeps track of which process is using which qubit at a given time. In general, a QMMU should take into account that the topology of a quantum memory determines what operations can be performed on which qubits, and thus allow processes to allocate qubits of a specific type upon request. An advanced QMMU could also feature algorithms to move qubits in the background — that is, without an explicit instruction from a process's routine — to accommodate an application's topology requirements while not trashing the qubits being used by other QNodeOS processes. Such a feature could prove crucial to increase the number of processes that can be using the quantum memory at the same time, and to enhance multitasking performances.

The QMMU exposes interfaces for three services:

- *Qubit allocation and deallocation* (interface ⑪ in Figure A.1): a running process can ask for one or more qubits, which, if available, are allocated by the QMMU, and their physical addresses are mapped to the virtual addresses provided by the requesting process.

- *Virtual address translation* (interface ⑫ in Figure A.1): before sending quantum instructions to the QDevice driver, the processor uses virtual qubit addresses specified in QASM to retrieve physical addresses from the QMMU, and then replaces virtual addresses with physical addresses in the instructions for the QDevice driver.
- *Qubit ownership transfer* (interface ⑬ in Figure A.1): qubits are only visible to the process that allocates them. However, in some cases, a process may wish to transfer some if its qubits to another one. A notable example is the quantum network process transferring an entangled qubit to the process that will use it.

References

[1] A. Dahlberg, M. Skrzypczyk, T. Coopmans, L. Wubben, F. Rozpędek, M. Pompili, A. Stolk, P. Pawełczak, R. Knegjens, J. de Oliveira Filho, R. Hanson, and S. Wehner. “A Link Layer Protocol for Quantum Networks”. In: *SIGCOMM*. ACM, 2019, pp. 159–173. DOI: 10.1145/3341302.3342070.

B QDevice Interface

The implementation of a QDevice depends on a number of factors. Most importantly, the physical signals that are fed to the quantum processing and networking device, and those that are output from the device, are specific to the nature of the device itself. Different qubit realizations require different digital and analog control. For instance, manipulating the state of a spin-based qubit (e.g., in a nitrogen-vacancy center processor) and that of an ultracold atom qubit (e.g., in a trapped ion processor) are two physical processes that vastly differ in a number of complicated ways.

For QNodeOS to be portable to a diverse set of quantum physical platforms, there needs to be a common *QDevice interface* that QNodeOS can rely on, and that each QDevice instance can implement as it is most convenient for the underlying quantum device. This interface need be quite general, to be able to express all quantum operations that different quantum devices might be capable of performing, and rather abstract, so that two different implementations of a well-defined qubit manipulation operation can be expressed with the same instruction on QNodeOS. Nevertheless, an interface that is too general could result in a high implementation complexity on the QDevice, as it might have to transform high-level instructions in a series of native operations on the fly. Other than complexity of implementation, a very high-level set of QDevice instructions might compromise the compiler's ability to optimize an application for a certain physical platform, as reported by Murali et al. [1].

Defining a set of instructions to express abstract quantum operations as close as possible to what different quantum physical platforms can natively perform is, to some extent, an open problem. While this is outside the scope of this work, we have made an effort to specify an interface which is a good compromise between generality and expressiveness. The QDevice interface is essentially a set of instructions that QNodeOS expects a QDevice to implement. To be precise, a QDevice might implement a subset of the interface, according to what native physical operations it can perform. The Host compiler must then have knowledge about the set of instructions implemented by the underlying QDevice, so that it can decompose instructions that are not natively supported.

Even though this interface does not impose any formal timing constraints, it is important to note that a QDevice implementation that tries to guarantee more or less determin-

Instruction	Description
`INI`	Initialize a qubit to default state
`SQG`	Perform a single-qubit gate
`TQG`	Perform a two-qubit gate
`AQG`	Perform a gate on all qubits
`MSR`	Measure a qubit in a specified basis
`ENT`	Attempt entanglement generation
`ENM`	Attempt entanglement and measure qubit
`MOV`	Move qubit state to another qubit
`SWP`	Swap the state of two qubits
`ESW`	Swap qubits belonging to two entangled pairs
`PMG`	Set pre-measurement gates

Table B.1: Summary of QDevice instructions defined in the QDevice interface. A specific QDevice might implement a subset of these, depending on the underlying quantum physical device and on other design constraints.

istic instruction processing latencies can prove more beneficial to the real-time requirements of QNodeOS. Particularly, it would be advisable to time-bound the processing time of operations whose duration is by nature probabilistic — most notably, those involving entanglement generation. Creating an entangled pair may involve a varying number of attempts. Sometimes, if the remote node becomes unresponsive for a period of time, the number of necessary attempts can increase by a large amount. Capping the number of attempts could, for instance, provide a more deterministic maximum processing latency for entanglement instructions, which in turn might help QNodeOS react more timely to temporary failures or downtime periods of remote nodes. Also, unbounded entanglement attempts affect the state of other qubits in memory, because of both passive decoherence and cross-qubit noise.

Table B.1 lists the complete set of instructions defined in the QDevice interface. Instructions can have operands, whose range of valid values depends on the underlying QDevice. For instance, an operand that specifies which qubit to apply an operation to can only have as many valid values as there are physical qubits in memory. Details for each instruction and its operands are given below.

Qubit initialization (`INI`). The `INI` instruction brings a qubit to the $|0\rangle$ state. On some physical platforms, single-qubit initialization is not possible, thus this instruction initializes all qubits to the $|0\rangle$ state.

Operand	Description
`qubit`	Physical address of the qubit to initialize, ignored on platforms where single-qubit initialization is not possible

Single-qubit gate (`SQG`). The `SQG` instruction manipulates the state of one qubit. The gate is expressed as a rotation in the Bloch sphere.

Operand	Description
`qubit`	Physical address of the qubit to manipulate
`axis`	Rotation axis, can be X, Y, Z or H (support is QDevice-dependent)
`angle`	Rotation angle (granularity and range are QDevice-dependent)

Two-qubit gate (`TQG`). The `TQG` instruction manipulates the state of two qubits. The gate is expressed as a controlled rotation, with one qubit being the control and the other one being the target.

Operand	Description
`qub_c`	Physical address of the control qubit
`qub_t`	Physical address of the target qubit
`axis`	Rotation axis, can be X, Y, Z or H (support is QDevice-dependent)
`angle`	Rotation angle (granularity and range are QDevice-dependent)

All-qubit gate (`AQG`). The `AQG` instruction manipulates the state of all available qubits. The gate is expressed as a rotation in the Bloch sphere.

Operand	Description
`axis`	Rotation axis, can be X, Y, Z or H (support is QDevice-dependent)
`angle`	Rotation angle (granularity and range are QDevice-dependent)

Qubit measurement (`MSR`). The `MSR` instruction measures the state of one qubit in a specified basis. A qubit measurement is destructive — that is — the qubit has to be reinitialized before it can be used again.

Operand	Description
`qubit`	Physical address of the qubit to measure
`basis`	Measurement basis, can be X, Y, Z, H (support is QDevice-dependent)

Entanglement generation (`ENT`). The `ENT` instruction performs a series of entanglement generation attempts, until one succeeds, or until a maximum number of attempts is reached (the behavior is QDevice-dependent).

Operand	Description
`nghbr`	Neighboring node to attempt entanglement with, if the local QDevice has multiple quantum links
`fid`	Target entanglement fidelity (granularity and range are QDevice-dependent)

Entanglement generation with qubit measurement (`ENM`). The `ENM` instruction performs a series of entanglement generation attempts followed by an immediate measurement of the local qubit, until one succeeds, or until a maximum number of attempts is reached (the behavior is QDevice-dependent).

Operand	Description
`nghbr`	Neighboring node to attempt entanglement with, if the local QDevice has multiple quantum links
`fid`	Target entanglement fidelity (granularity and range are QDevice-dependent)
`basis`	Measurement basis, can be X, Y, Z, H (support is QDevice-dependent)

Qubit move (`MOV`). The `MOV` instruction moves the state of one qubit to another qubit. A qubit move renders the state of the source qubit undefined, and the qubit has to be reinitialized before it can be used again.

Operand	Description
`qub_s`	Physical address of the source qubit
`qub_d`	Physical address of the destination qubit

Qubit swap (`SWP`). The `SWP` instruction swaps the state of two qubits.

Operand	Description
`qub_1`	Physical address of the first qubit
`qub_2`	Physical address of the second qubit

Entanglement swap (`ESW`). The `ESW` instruction is effectively a Bell state measurement executed on two qubits — each of which belongs to an entangled pair shared between the local node and a remote counterpart — resulting in an entangled pair shared between the two remote nodes. The outcome of the measurement is typically used by the quantum network stack to issue appropriate Pauli corrections on one of the end nodes in order to deliver the desired state.

Operand	Description
`qub_1`	Physical address of the first qubit
`qub_2`	Physical address of the second qubit

Pre-measurement gates setting (`PMG`). The `PMG` instruction allows for a set of (up to) 3 rotations to be performed before a qubit measurement (`MSR` or `ENM`). If the axis of the second rotation is orthogonal to the axis of the first and the third rotation, these gates can be used to perform a qubit measurement in an arbitrary basis, given that most likely a QDevice can natively measure in a limited set of bases.

Operand	Description
`axes`	Combination of orthogonal axes to use for the three successive rotations, can be X–Y–X, Y–Z–Y and Z–X–Z (support is QDevice-dependent)
`ang_1`	Rotation angle of the first gate, relative to the first axis in `axes` (granularity and range are QDevice-dependent)
`ang_2`	Rotation angle of the second gate, relative to the second axis in `axes` (granularity and range are QDevice-dependent)
`ang_3`	Rotation angle of the third gate, relative to the third axis in `axes` (granularity and range are QDevice-dependent)

References

[1] P. Murali, N. M. Linke, M. Martonosi, A. J. Abhari, N. H. Nguyen, and C. H. Alderete. "Full-Stack, Real-System Quantum Computer Studies: Architectural Comparisons and Design Insights". In: *ISCA*. ACM, 2019, pp. 527–540. DOI: 10.1145/3307650.3322273.

C Implementation of the Quantum Physical Layer

We provide additional details concerning the implementation of the quantum physical layer used for our experiments.

Single qubit gates. At the physical layer, we implement real-time rotations around the X and Y axes of the qubit Bloch sphere, using a resolution of $\pi/16 = 11.25°$. That is, the upper layer can request any rotation that is a multiple of $\pi/16$ around either the X or Y axis. The different rotations are performed using Hermite-shaped pulses (as described in Ref. [1]) of calibrated amplitude. The choice of X(Y) rotation axis is implemented using the I(Q) channel of the microwave vector source.

While supported on QNodeOS, our physical layer currently does not implement Z-axis rotations. Such rotations around the Z axis could be implemented by virtual rotations of the Bloch sphere: a π pulse around the Z axis is equivalent to multiplying future I and Q voltages by -1. By keeping track of the accumulated Z rotations, and by adjusting I and Q mixing accordingly, one can perform effective Z rotations with very high resolution and virtually no infidelity. The AWGs currently in use have the required capabilities, and the implementation of said Z gates is planned for the near future.

Clock sharing and AWG triggering over longer distances. One of the technical challenges of realizing a large scale quantum network is synchronizing equipment at the physical layer across nodes. The synchronization is required to generate entanglement — the photons from the two nodes need to arrive at the same time at the heralding station (compared to their duration, 12 ns for NV centers in bulk diamond samples); failing to do so would reduce (or even remove) their indistinguishability, which is required to establish long-distance entanglement [1]. Our two nodes are located in a single laboratory, on the same optical table, approximately 2 m apart. This allows for some simplifications, for the purpose of demonstrating entanglement delivery using a network stack, which would not be possible over longer distances. Specifically:

1. We use a single laser — the client's — to excite both nodes, as in Ref. [1]. Over longer distances, one would need to phase-lock the excitation lasers at the two nodes to ensure phase-stability of the entangled states.
2. The Device Controllers (ADwin Pro II microcontroller [2]) are triggered every 1 µs by the same signal generator, advancing the state machine algorithm that implements the physical layer. This ensures that the two microcontrollers have a common shared clock. Over longer distances, one could use existing protocols (and commercially-available hardware) to obtain a shared clock [3], and use that to trigger the microcontrollers.
3. The two AWGs need to be triggered to play entanglement attempts. In our implementation, one device controller — the server's — triggers both AWGs. This ensures that the triggering delay between the two AWGs is constant, and we can therefore calibrate it out. Triggering the AWGs with two independent microcontrollers would result in jitter (realistically on the order of nanoseconds). Over larger distances, one could derive — from the shared clock — a periodic trigger signal that is gated by the microcontroller at each node. In this way the microcontroller can decide whether the AWG will be triggered on the next cycle, but the accuracy of the trigger's timing will be derived from the shared clock between the nodes, rather than from the microprocessor.
4. The phase stabilization scheme we use, developed in Ref. [1], is designed to work at a single optical frequency (in our case, the 637 nm emission frequency of the NV center). Over longer distances, conversion of the NV center photons to the telecom band will be necessary to overcome photon loss. The phase stabilization scheme will therefore need to be adapted to new optical frequencies used.

For reference, our client (server) is based on node Charlie (Bob) of the multi-node quantum network presented in Ref. [1].

NV center resonance control. The two quantum network nodes use different techniques to control the resonance of their NV centers (see Ref. [1] for implementations details). The server uses an off-resonant charge randomization strategy: when its NV center is not on resonance (it does not pass the charge and resonance check), it can apply an off-resonant (green, 515 nm) laser pulse to shuffle the charge environment and probabilistically recover the correct charge and resonance state. The server cannot get *stuck* in a non-resonance state: in a few tens of failed CR checks and green laser pulses (overall less than 1 ms) the NV center will be in resonance again.

The client, which needs to be tuned in resonance with the other node, uses a resonant strategy. When in the wrong charge state (zero counts during CR check), it applies a resonant laser pulse (yellow, 575 nm, NV^0 zero-phonon line) to go back to NV^-. To bring NV^- in resonance with the necessary lasers, it adjusts a biasing voltage applied to the diamond sample, which shifts the resonance frequencies. This process is mostly automated. However, occasional human intervention is still required when the resonance frequencies of the NV center shift too far — for example due to a charge in the vicinity of the NV center changing position in the lattice — for the automatic mechanism to find its way back. Periods of inactivity in entanglement generation are due to the jumps in the client's NV optical transitions, which then require manual optimization of the laser frequencies and/or the diamond biasing voltage — depending on the magnitude of the frequency shift, it requires tens of seconds to a few minutes to recover the optimal resonance condition.

References

[1] 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”. In: *Science* 372.6539 (2021), pp. 259–264. DOI: 10.1126/science.abg1919.

[2] *ADwin-Pro II*. Jäger GmbH. URL: https://www.adwin.de/us/produkte/proII.html (visited on Feb. 28, 2023).

[3] *The White Rabbit Project*. CERN. URL: https://white-rabbit.web.cern.ch/ (visited on Feb. 28, 2023).

D
Raw Distributions of RTT Measurements

We plot in Figure D.1 the raw distributions of RTT delays measured in Chapter 6. As already mentioned, the RTT measurements for the “Bypass servers” configuration can be approximated by a single-mode Gaussian distribution, with small standard deviation and negligible outliers. Such a clean distribution is an indicator of the stable performance of the real-world classical network used in our experiments and of the fairly constant traffic on such network. On the other hand, measurements for the “No MAC” and “Poly1305-AES” configurations are better approximated by bimodal Gaussian distributions, with negligible differences in mean and variance of each mode across the two configurations. These more stark variations in RTT for these two configurations are most likely to be attributed to some caching behavior in the packet processing pipeline. Nevertheless, the overall mean RTT for each configuration is approximately equal to the mean of the strongest mode. The following tables report mean and standard deviation for all configurations, and for all modes of the distributions — where these are bimodal.

Server		**RTT mean** [μs]		**RTT std** [μs]	
configuration	**Scope**	**Pl. 12**	**Pl. 1200**	**Pl. 12**	**Pl. 1200**
Bypass servers	Overall	3657	3881	23	22
No MAC	Overall	13 821	13 993	3142	2876
	First mode	14 036	14 089	797	767
	Second mode	11 251	11 400	446	397
Poly1305-AES	Overall	14 387	13 959	5044	2867
	First mode	14 099	14 015	861	729
	Second mode	11 300	11 442	398	356

Table D.1: Mean and standard deviation of RTT for different configurations of the DOA servers and for all modes of the analyzed distributions, for payloads of 12 bytes (Pl. 12) and 1200 bytes (Pl. 1200).

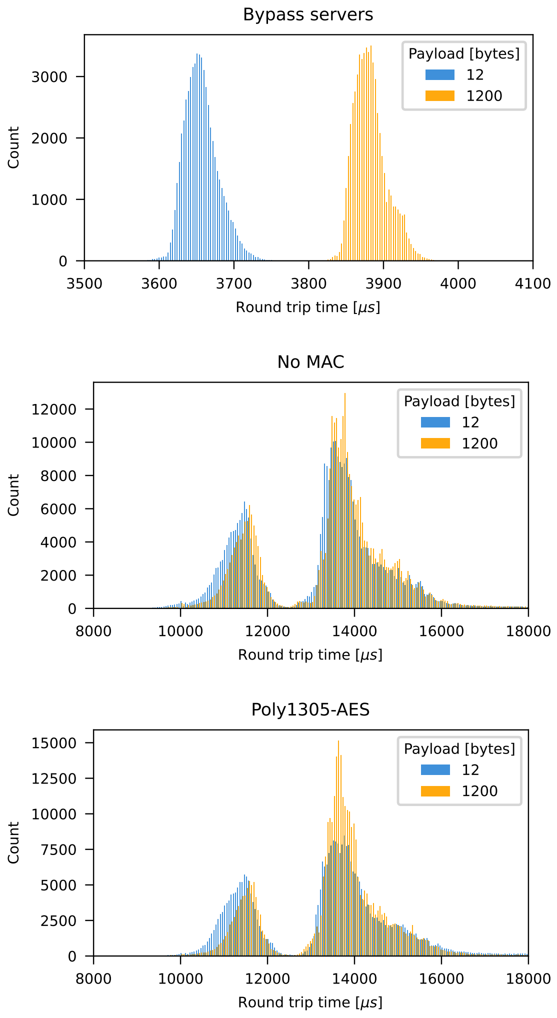


Figure D.1: Distribution of RTT measurements for the “Bypass servers”, “No MAC” and “Poly1305-AES” configurations, for message payloads of 12 and 1200 bytes. Each histogram consists of 200 bins.

Glossary

API application programming interface
ASIC application-specific integrated circuit
AWG arbitrary waveform generator

CK create and keep
CPLD Complex Programmable Logic Device
CPU central processing unit
CR check charge and resonance check

DIO digital input/output
DOA data origin authentication
DQP distributed queue protocol

EMU entanglement management unit

FPGA field-programmable gate array

GPS Global Positioning System

HAL hardware abstraction layer

ICMP internet control message protocol
IP internet protocol
IPC inter-process communication
IPv4 Internet Protocol version 4
ITS information-theoretic security

MAC message authentication code
MD measure directly
MDI-QKD measurement-device independent quantum key distribution
MHP midpoint heralding protocol

NL network layer
NV nitrogen-vacancy

OS operating system
OSI Open System Interconnect

PCB process control block
Poly1305-AES Poly1305-AES message authentication code

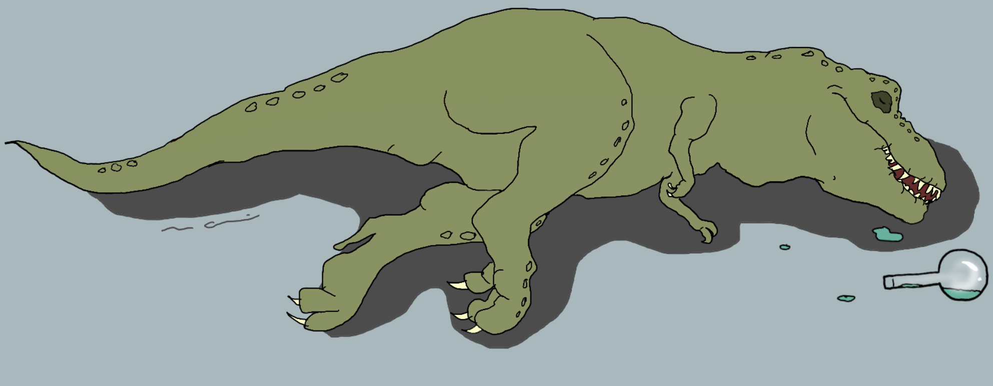
QDevice physical-layer quantum device
QEGP quantum entanglement generation protocol
QKD quantum key distribution
QMMU quantum memory management unit
QNodeOS operating system for quantum network nodes
QNP quantum network protocol
QNPU quantum network processing unit

RTT round-trip time

SDK software development kit
SDN software-defined networking
SPI Serial Peripheral Interface

TCP Transmission Control Protocol
TCP/IP Internet protocol suite
TDMA time-division multiple access

VMAC VMAC message authentication code



ISBN 978-94-6384-457-4