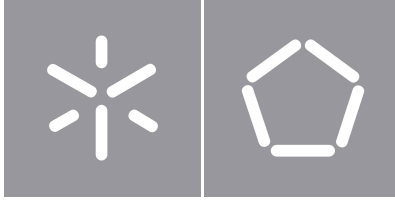


University of Minho
School of Engineering

Pedro Paulo Queirós Tavares

**SDN Controller for
Quantum Key Distribution
in 6G Networks**



University of Minho
School of Engineering

Pedro Paulo Queirós Tavares

**SDN Controller for
Quantum Key Distribution
in 6G Networks**

Masters Dissertation
Master's in Informatics Engineering

Dissertation supervised by
António Costa

Copyright and Terms of Use for Third Party Work

This dissertation reports on academic work that can be used by third parties as long as the internationally accepted standards and good practices are respected concerning copyright and related rights.

This work can thereafter be used under the terms established in the license below.

Readers needing authorization conditions not provided for in the indicated licensing should contact the author through the RepositóriUM of the University of Minho.

License granted to users of this work:



CC BY

<https://creativecommons.org/licenses/by/4.0/>

Acknowledgements

First and foremost, I would like to thank my supervisor, Prof. António Costa, for accepting me as his master's student and for the invaluable guidance and support during this journey.

I would also like to thank José Cunha and Optare Solutions for the opportunity to work on this topic, which significantly improved my knowledge in the field.

Special thanks to Lluís Gifre, who developed most of TeraflowSDN, for his technical expertise, enlightening advices and also for guiding me through the right path.

I also have to extend my sincerest expression of gratitude to my family, friends, and loved ones who have stood by my side since the beginning of this project. Their support and encouragement meant much to me, as did the moments when they helped me take my mind off these thoughts during my free time. This really made it possible for me to do my best for this thesis.

Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

University of Minho, Braga, october 2024

Pedro Paulo Queirós Tavares

Abstract

In future 6G networks, secure communications are expected to use end-to-end quantum encryption. For that, efficient and scalable Quantum Key Distribution (QKD) is required. Since 6G networks already rely on the Software Defined Networking (SDN) paradigm for flexible network management, one possible solution is to integrate quantum key distribution functionality in a SDN Controller.

In this dissertation, a new Quantum Key Distribution Network Controller is proposed, designed to smoothly integrate into the existing SDN Controller, named TeraFlowSDN , allowing for a management of the new predicted QKD Nodes. This involved a study of the state of art, a conceptual view of the architectures of both the current TeraflowSDN and the newly proposed QKD Network, a possible integration of the new QKD Network Controller into TeraflowSDN and a validation of the new system.

QKD Network's architecture is a newly proposed architecture to bring QKD into our current systems. It contains components such as QKD Nodes, QKD SDN Orchestrator and QKD SDN Controller to enable the QKD communications. The QKD SDN Controller is the main component of study from the QKD Network since it's the one which will be used to integrate into the TeraflowSDN's architecture. TeraflowSDN's architecture is made up of a few modules, each one with their own functionality, such as displaying a web interface, communicating with external infrastructure, computing path, managing services, storing data and monitoring.

The integration presented in this dissertation is still a preliminary implementation which was done before having real QKD Nodes (Infrastructure) available. Because of this, validation was done on a virtual environment such as loading a file descriptor and creating QKD Devices, QKD Services (Direct and Virtual) and QKD Apps into the QKD SDN Controller.

Keywords 6G, QKD SDN Controller, TeraflowSDN, Quantum Key Distribution, Software Defined Networking

Resumo

Nas futuras redes 6G, espera-se que as comunicações seguras utilizem encriptação quântica de ponta a ponta. Para isso, é necessária uma Distribuição de Chaves Quânticas (do inglês *Quantum Key Distribution*, QKD) eficiente e escalável. Uma vez que as redes 6G já dependem do paradigma de Redes Definidas por Software (do inglês *Software Defined Networking*, SDN) para a gestão flexível da rede, uma solução possível é integrar a funcionalidade de distribuição de chaves quânticas num Controlador SDN.

Nesta dissertação, é proposto um novo Controlador de Rede de Distribuição de Chaves Quânticas, concebido para se integrar de forma fluida no Controlador SDN existente, denominado TeraflowSDN, permitindo a gestão dos novos Nós QKD previstos. Isto envolveu um estudo do estado da arte, uma visão conceptual das arquiteturas tanto do atual TeraflowSDN como da nova Rede QKD proposta, uma possível integração do novo Controlador de Rede QKD no TeraflowSDN e uma validação do novo sistema.

A arquitetura da Rede QKD é uma arquitetura recentemente proposta para trazer o QKD para os nossos sistemas atuais. Contém componentes como Nós QKD, Orquestrador SDN QKD e Controlador SDN QKD para permitir as comunicações QKD. O Controlador SDN QKD é o principal componente de estudo da Rede QKD, uma vez que será utilizado para se integrar na arquitetura do TeraflowSDN. A arquitetura do TeraflowSDN é composta por vários módulos, cada um com a sua própria funcionalidade, tais como apresentar uma interface web, comunicar com infraestruturas externas, calcular caminhos, gerir serviços, armazenar dados e monitorizar.

A integração apresentada nesta dissertação ainda é uma implementação preliminar, que foi realizada antes da existência de Nós QKD (Infraestrutura) reais disponíveis. Por causa disso, as validações foram executadas num ambiente virtual, tais como carregar um descritor de ficheiro e criar Dispositivos QKD, Serviços QKD (Diretos e Virtuais) e Aplicações QKD no Controlador SDN QKD.

Palavras-chave 6G, Controlador SDN QKD, TeraflowSDN, Distribuição de Chaves Quânticas, Redes Definidas por Software

Contents

- 1 Introduction 1**
 - 1.1 Context 1
 - 1.2 Motivation 2
 - 1.3 Objectives 3
 - 1.4 Contributions 3
 - 1.5 Optare Solutions 3
 - 1.6 Thesis Structure 4

- 2 State of the Art 5**
 - 2.1 6th Generation of Wireless Technology 5
 - 2.1.1 What led to the development of 6G 5
 - 2.1.2 Comparison between 5G and 6G 6
 - 2.2 Quantum Topics 7
 - 2.2.1 Quantum-safe methods 7
 - 2.2.2 Quantum Key Distribution 8
 - 2.3 Software Defined Networking 9
 - 2.3.1 Network Function Virtualization 10
 - 2.3.2 Software Defined Networking Controllers 11
 - 2.4 Quantum Key Distribution - Software Defined Networking 14
 - 2.4.1 Architecture 15
 - 2.4.2 Interfaces 18
 - 2.5 Final remarks 19

- 3 Concept and Design of QKD into TeraflowSDN 20**
 - 3.1 Advantages of TeraflowSDN 20
 - 3.2 TeraflowSDN Architecture 21

3.3	How QKD Network works	23
3.4	QKD network Architecture	24
3.4.1	QKD Node	26
3.4.2	QKD Network Controller	35
3.4.3	QKD Orchestrator	37
3.5	Summary	38
4	Implementation	39
4.1	Introduction	39
4.2	Development Phases	39
4.2.1	Phase 1: Integration Planning	39
4.2.2	Phase 2: Simulation of QKD Nodes	40
4.2.3	Phase 3: Analysis and integration of components into TeraflowSDN	40
4.2.4	Phase 4: Deployment in a realistic environment	41
4.2.5	Phase 5: Test and Evaluation	41
4.3	Software Components	41
4.3.1	QKD Network Topology	41
4.3.2	SBI	42
4.3.3	QKD Network Control & Mgmt.	43
4.3.4	QKD Application Register	44
4.3.5	QKD Network Monitoring	46
4.3.6	Path Computation Engine	46
4.3.7	NBI	46
4.3.8	WebUI (Extra component)	46
4.4	Summary	47
5	Testing and Evaluation	48
5.1	Introduction	48
5.2	Use/Test Cases	48
5.2.1	Home Page (TeraflowSDN Working)	49
5.2.2	Load Topology from SBI into TFS using a File Descriptor	49
5.2.3	Create New QKD Device	50
5.2.4	Create New Direct/Physical Service	52

5.2.5	Create New Virtual Service	54
5.2.6	Create New External App via REST API	56
5.3	Summary	56
6	Conclusions and future work	57
6.1	Conclusions	57
6.2	Prospect for future work	58
I	Appendices	66
A	The SBI as defined in ETSI QKD 015 v2.1.1	67
B	QKD Network topology stored in TeraflowSDN	69
C	Database schema for new App Component	71
D	Example Network Topology in JSON format	72

List of Figures

- 1 TeraflowSDN Scenario 13
- 2 QKD network architecture 15
- 3 Use case of SDN orchestrator and SDN controllers for QKD network and Optical Network 16
- 4 Depiction of an QKD SDN network showing a set of QKD nodes 17
- 5 TeraflowSDN Architecture 21
- 6 Network Elements 23
- 7 Network Provision for Quantum-Secure Communications 23
- 8 QKD Network Architecture 25
- 9 Minimum QKD Node 26
- 10 Complex QKD Node 27
- 11 QKD Module 28
- 12 Scheme of key relay from point to point (Case 1) 29
- 13 Scheme of key relay from point to point (Case 2) 30
- 14 Scheme of key relay with XORs uniformly processed at destination node (Case 3) 31
- 15 Scheme of key relay with XORs collected at one centralized node (Case 4) 32
- 16 KMS (Key Management System) 33
- 17 QKD Node Controller 34
- 18 QKD Network Controller Architecture 36
- 19 QKD Orchestrator 38
- 20 Sequence diagram for the dynamic creation of a virtual (multi-hop) link 44
- 21 Sequence diagram of applications registration in an SDN QKD network 45
- 22 Homepage 49
- 23 Homepage - Toplogy loaded 50
- 24 Devices 50
- 25 Add new Device 51

26	Device details	52
27	Create direct/physical Service	52
28	Services after direct/physical service	53
29	Devices after direct/physical service	53
30	Device details after direct/physical service	53
31	Create Virtual Service	54
32	Services after virtual service	54
33	Devices after virtual service	55
34	Device details after virtual service	55
35	Apps after virtual service	56
36	Request external App	56
37	Apps after external app	56

List of Tables

- 1 Differences between 5G and 6G 7
- 2 Comparison of SDN and NFV concept 11
- 3 Comparison between TeraFlowSDN and OpenDayLight 13

Acronyms

ACL Access Control List.

AI Artificial Intelligence.

API Application Programming Interface.

BBF Broadband Forum.

BCI Brain-Computer Interactions.

BGP Border Gateway Protocol.

BGP-LS Border Gateway Protocol Link State.

BSS Business Support Systems.

CLI Command-line interface.

CRAS Connected Robotics and Autonomous Systems.

CRUD Create, read, update and delete.

CTTC Centre Tecnològic de Telecomunicacions de Catalunya.

DLT Distributed Ledger Technologies.

DWDM Dense wavelength-division multiplexing.

eMBB Enhanced Mobile BroadBand.

ETSI European Telecommunications Standards Institute.

gNMI gRPC Network Management Interface.

GUI Graphical User Interface.

HDD Hard Disk Drive.

HTTP Hypertext Transfer Protocol.

IBM International Business Machines.

IEC International Electrotechnical Commission.

IEEE Institute of Electrical and Electronics Engineers.

IETF Internet Engineering Task Force.

IoE Internet of Everything.

IoT Internet of Things.

IP Internet Protocol.

IPv4 Internet Protocol version 4.

ISO International Organization for Standardization.

ITU-T International Telecommunication Union Telecommunication.

JSON JavaScript Object Notation.

KMA Key Management Agent.

KMS Key Management System.

KSA Key Supply Agent.

MANO Management and Orchestration.

MEC Multi-access Edge.

MPLS Multiprotocol label switching.

NBI NorthBound Interface.

NEC Nippon Electric Company.

NETCONF Network Configuration Protocol.

NF Network Function.

NFV Network Functions Virtualization.

NIC Network Interface Controller.

NMS Network Management System.

ODL OpenDayLight.

ONAP Open Network Automation Platform.

ONF Open Networking Foundation.

ONOS Open Network Operating System.

OSS Operations Support Systems.

OTP One-Time Password.

P4 Programming Protocol-independent Packet Processors.

QKD Quantum Key Distribution.

QKDN Quantum Key Distribution Network.

RAM Random Access Memory.

REST Representational State Transfer.

RESTCONF Representational State Transfer Configuration.

RPC Remote Procedure Call.

SBI SouthBound Interface.

SDN Software Defined Networking.

SDQNC Software Defined Quantum Network Controller.

SNMP Simple Network Management Protocol.

SR-TE Segment Routing Traffic Engineering.

SSH Secure Socket Shell.

TAPI Transport Application Programming Interface.

TFS TeraflowSDN.

TLS Transport Layer Security.

TSP Telecommunications Service Provider.

UI User Interface.

URL Uniform Resource Locator.

URLLC Ultra-reliable, Low-latency Communications.

UUID Universally Unique Identifier.

vCPU Virtual Central Processing Unit.

VM Virtual Machine.

VNF Virtual Network Functions.

VPN Virtual Private Network.

XML Extensible Markup Language.

XOR Exclusively-OR.

XR Extended Reality.

YANG Yet Another Next Generation.

Chapter 1

Introduction

This chapter has the aim to introduce and give context about the major theme of this dissertation. The goal is to explain the problem, exposing the motivation behind it, establishing the objectives, contributions and its structure.

1.1 Context

Nowadays, we live in the era of the 5th Generation of Wireless Technology, 5G, however, some of us might have already heard something about the new 6th Generation of Wireless Technology, 6G. Even though it's still a few years away, some industry stakeholders are already making preparations and a few advancements in order to be prepared for this new era with new protocols and ideas. Until now, there are plenty of ideas and discussions about defining specs and standards behind the scenes. 6G is a conceptual wireless network technology that not only will succeed 5G but also power immersive, ubiquitous, and sensory digital experiences on a massive scale and also bring a self-sustaining connectivity by harvesting energy from the ambient.

This new 6G era brings with it the concept of quantum communications, where the domain Quantum key distribution, QKD, comes along. QKD is a secure communication method for exchanging encryption keys only known between shared parties. It uses properties found in quantum physics to exchange cryptographic keys in such a way that is provable and guarantees security. The ones who will support these QKD communications are called QKD Nodes and they connect to each other using QKD Links.

Back in the 4G era, a new approach to network management appeared. This new approach, called Software Defined Networking, SDN, is not tied to a wireless generation technology and became more useful with each generation. In the 5G era it was pretty useful along with the virtualization, and, in the 6G era it seems that it will be, also, pretty useful along with this new QKD concept.

SDN is a network architecture approach that enables the network to be intelligently and centrally con-

trolled. By separating the control plane from the data plane, it benefits from a more centralized management and a more flexible network configuration, allowing dynamic adjustments to traffic flow and policies without requiring changes to each of the underlying hardware. These changes are made by the controller which directly communicates with its underlying hardware infrastructure (such as switches). As a result, network operators can respond more quickly to changing demands and optimize resource usage effectively.

The **Opensec** project is an initiative focused on enhancing the security, trust, and management of 6G networks, which are expected to be open, disaggregated, and complex. The project is divided into three key sub-projects:

- 6G-OPENSEC_KEYS [1]: This sub-project focuses on revolutionizing 6G network security by integrating QKD technology.
- 6G-OPENSEC_SECURITY [2]: This sub-project aims to create an intelligent and autonomous security management system for 6G network slices.
- 6G-OPENSEC_TRUST [3]: This sub-project seeks to design and implement a Trust manager that will assess and manage the trustworthiness of service providers in a multi-vendor 6G environment.

Specifically, this dissertation was developed on the context of the project 6G-OPENSEC-KEYS which proposes to adopt flexible and efficient software-controllable QKD solutions using SDN technology to ensure security in 6G networks. An integration of QKD technologies will also be promoted in a sustainable manner considering the deployed network infrastructure and a better use of network resources.

1.2 Motivation

This new 6G era brings the QKD Nodes to support the QKD communications. In order for the QKD Nodes to properly function and work on an intelligent and centrally controlled QKD network, an approach should be used known as SDN. This approach after being adapted to control the new QKD Network, will be able to bring this to an higher level by allowing new management activities such as monitoring, directing traffic, controlling QKD Nodes and others numerous new possibilities.

In the market, there are a great number of SDN Controllers such as TeraflowSDN and OpenDayLight. Each one is unique and can communicate with a variety of infrastructures' types. For example, both of them support OpenConfig protocol used to communicate with Switches, allow the creation of VPNs and the possibility to have an orchestrator above to orchestrate their network beneath.

The motivation for this dissertation is to provide support for QKD within current SDN controllers.

1.3 Objectives

In this dissertation, the main objective is to **have an SDN Controller which is able to control the new QKD Nodes**. Given that this dissertation was conducted within the scope of the *Opensec* project, it was decided to extend an existing SDN controller in order to meet the project's goals.

Throughout this dissertation several points will be addressed by the following order:

1. A study of the state of art was done to explore concepts like 6G, QKD, SDN. Furthermore it was also essential to acquire an idea of the existing SDN controllers.
2. Concept of QKD, in-depth exploration of the entire QKD network and study its integration into an existing SDN Controller.
3. Design of possible architecture for the integration of QKD into an SDN Controller.
4. Implementation of the designed architecture and description of each phase of development in detail.
5. Utilization and validation of the system. Involves practical deployment, testing and evaluation to confirm its effectiveness.

1.4 Contributions

Along with this dissertation, a related paper for the 15th International Conference on Network of the Future (NoF 2024 [4]) was carried out and it was already accepted and will be part of the IEEE Xplore database. This paper, similarly to this dissertation, talks about a new Quantum Key Distribution Network Controller which was proposed, designed to smoothly integrate into the existing TeraFlowSDN. Additionally, the source code of the implementation will be available on TeraflowSDN's repository [5].

This dissertation aims to develop a key part of a project focused on revolutionizing security in 6G networks to address the challenges of open, disaggregated technologies and the threat of quantum computing.

1.5 Optare Solutions

This work was done in context of an internship at Optare Solutions from September 2023 until August 2024.

Optare Solutions [6] is a consultant firm, based in Vigo (Spain), working exclusively for telecommunications operators worldwide, with high specialization in Operations Support Systems (OSS) and Business Support Systems (BSS).

1.6 Thesis Structure

This dissertation is structured as follows: it begins with a brief introduction to the theme, presenting its motivations and objectives. This is followed by a state of the art, which presents useful background concepts, with a focus on 6G, QKD, and SDN. Next, it covers the concepts, explaining the design of the SDN in use (TeraflowSDN) and the entire QKD network in detail. Succeeded by a discussion on the integration of the new QKD Controller into the existing SDN Controller (TeraflowSDN). After that, it explores the applications, detailing how the new integration works and the tests conducted. Finally, the dissertation concludes with possible modifications and suggestions for future work.

Chapter 2

State of the Art

In this chapter, the fundamental concepts and some information on related works are presented. It begins with a section dedicated to the 6th Generation of Wireless Networks, where a comparison with the current 5G generation is also made. Next, in section 2, the concepts related to quantum key distribution are introduced. The third section is dedicated to software-defined networks and SDN controllers, and the fourth section to quantum key distribution in the SDN context. The chapter ends with a section dedicated to related works.

2.1 6th Generation of Wireless Technology

6th Generation of Wireless Technology (6G) is still in the early stages of development. It is expected to be a significant advancement over its predecessor, 5G, introducing new capabilities far beyond the limits of 5G.

The following sections present an explanation of the reasons behind the development of 6G, including the technologies and applications it will involve. Additionally, there is a comparison of 6G and 5G complemented with a table supporting it.

2.1.1 What led to the development of 6G

The article written by *Walid Saad et al.* [7], the authors mention a lot of improvements from this new transition from the old 5G into the new 6G. Previously, the main focus of wireless networks was to guarantee the best mobile broadband (eMBB) and high data speeds. However, nowadays, the most important part is guaranteeing the best performance for ultra-reliable low-latency communications (URLLC). This transition of the focus is due to the increasing use of Internet of Everything (IoE) paradigm, a concept where people, things, and places are interconnected. New applications from IoE, such as smart city IoE and extended reality (XR) services, suggest that even though 5G allowed these applications to exist, it still

needs improvement to fully exploit all their possibilities.

The main reason for 6G to start being developed was the demand for a wireless system which could meet specific performance requirements of IoE applications. New advancements in fields such as computing, artificial intelligence, sensing and wireless devices are pushing it forward.

5G created the base for services that lead to the demand of 6G. Applications and technologies such as: Connected Robotics and Autonomous Systems (CRAS), systems where robots and autonomous devices are interconnected and can operate together; Wireless Brain-Computer Interactions (BCI), technologies that allow direct communication between the brain and external devices without the need for wires; Multisensory Extended Reality (XR) Applications, applications that are expected to provide multisensory experiences, combining visual, auditory, and possibly other sensory inputs; Blockchain and Distributed Ledger Technologies (DLT), technologies that provide secure and decentralized ways to record and verify transactions and data. All these applications and technologies require URLLC which is the main focus of 6G.

2.1.2 Comparison between 5G and 6G

In the article written by *Shuping Dang et al.* [8], authors say that 6G is predicted to improve the security of devices, secrecy and privacy through modern technologies since the traditional methods for encryption are considered insecure in the era of big data and artificial intelligence. Furthermore, small steps towards better energy efficiency, more intelligence, lower costs, and customer-tailored design are also expected. In terms of energy efficiency, advances in green communications and energy harvesting will be useful. Even though 6G seems to revolutionize many aspects in current networks, there are some points where it won't be possible such as in the spectral efficiency. 6G may not undergo any significant change because the Shannon bound sets a theoretical maximum on the data rate that can be transmitted over a communication channel for a given bandwidth and signal-to-noise ratio, without error. This limit is a fundamental principle in information theory. By other words, 5G has already approached this limit, as a result, further improvements in spectral efficiency for 6G are likely to be incremental rather than revolutionary.

The table 1 represents some of the differences between 5G and 6G. For more detailed differences check the related paper written by *Samar Elmeadawy et al.* [9].

Table 1: Differences between 5G and 6G, Source: [9]

Characteristic	5G	6G
Operating frequency	3 - 300 GHz	upto 1 THz
Uplink data rate	10 Gbps	1 Tbps
Downlink data rate	20 Gbps	1 Tbps
Spectral efficiency	10 bps/Hz/m ²	1000 bps/Hz/m ²
Reliability	10 ⁵	10 ⁹
Maximum mobility	500 km/h	1000 km/hr
U-plane latency	0.5 msec	0.1 msec
C-plane latency	10 msec	1 msec
Processing delay	100 ns	10 ns
Traffic capacity	10 Mbps/m ²	1 - 10 Gbps/m ²
Localization precision	10 cm on 2D	1 cm on 3D
Satellite integration	No	Fully
AI integration	Partially	Fully
XR integration	Partially	Fully

2.2 Quantum Topics

The article written by *Yuan Chao et al.* [10] states that the increased computational power, derived from the development of quantum computers, threatens conventional cryptosystems. This happens because the new quantum computers can solve algorithms much faster compared to conventional computers which implies deciphering keys much faster bringing a problem to current security.

These threats are becoming more evident and quantum-safe measures must be considered. In this section, it will be highlighted the different quantum-safe methods which can be applied to increase security followed by a detailed explanation on how quantum key distribution comes in hand.

2.2.1 Quantum-safe methods

Nowadays, there are two main quantum-safe methods mentioned by *Yuan Cao et al.* [10] named post-quantum cryptography and quantum cryptography. Post-quantum cryptography is an approach to classical cryptography which is designed to resist attacks by quantum computers. It's based on complex

mathematical formulas which are considered too hard even for a quantum computer. This method consists of algorithms such as code-based [11], hash-based [12], lattice-based [13] and multivariate [14] that have been proven safe against the known quantum attacks. On the other hand, quantum cryptography is a completely different way of thinking that uses laws of physics that takes advantage of atoms and molecules in order to secure a safer communication.

From these two methods, post quantum cryptography has been considered capable of working together with the current cryptographic systems and also have good secret key rates over long distances which are known as quantum-resistance but could be exposed to future unknown algorithms. On contrast, quantum cryptography implies using principles of quantum physics which requires physical changes in current systems.

2.2.2 Quantum Key Distribution

Quantum Key Distribution (QKD) is defined by *Yuan Cao et al.* [10] as a symmetric secret key negotiation protocol capable of maintaining information-theoretic security. The quantum cryptography method requires the use of Quantum Key Distribution (QKD) protocols in order to safely exchange cryptographic keys. The security is based on fundamental properties of particles such as Heisenberg's uncertainty principle [15] and the no cloning theorem [16].

Some other authors, such as *Chonggang Wang et al.* [17], reinforce the idea that QKD is a secure communication method which allows two parties to securely exchange cryptographic keys over a quantum channel. These QKD protocols use the principles of quantum mechanics, particularly the non-cloning theorem, which states that quantum states cannot be copied, guaranteeing that any attempt to intercept the communication will be detected, avoiding eavesdropping.

Yuan Cao et al. [10] talks about QKD networks, highlighting Software-Defined Networking (SDN) as an important concept for the network layer. SDN improves efficiency in controlling and managing a whole network from a centralized platform that is flexible and programmable which will benefit the management of future QKD Networks.

Additionally, there are ongoing standardization efforts in QKD by groups like ETSI, ITU-T, ISO/IEC, IETF, and IEEE. Allowing for many various application scenarios on QKD networks, including finance, government, cloud services, critical infrastructures, healthcare, and space and mobile applications. Specific applications include securing links in financial institutions, improving voting systems, securing data backups, and protecting critical infrastructures such as energy grids and aeronautical telecommunication networks.

2.3 Software Defined Networking

The Software Defined Networking (SDN) is defined by *Diego Kreutz et al.* [18] as a new model of networking that has been designed to improve the complexity of managing and controlling the complex IP networks. In the traditional network setups, it is not only hard to do configuration but also reconfiguration. In fact, data and control planes are tightly integrated thus restricting any room for innovative thinking and flexibility on the part of the two parties.

This new SDN model makes possible the detachment of the control logic from the underlying data plane (switches as well as routers). This facilitates centralized management and programmability of networking devices. By doing this, networks can support efficient and dynamic resource management, while at the same time allowing development of recent network applications. For instance, SDN facilitates network virtualization where multiple virtual networks are built on one physical infrastructure shared among them.

Rashid Amin et al. [19] states that these switches and routers are also able to see and transfer network traffic flows according to the rules set by the controller. This way of doing things makes network control and management less burdensome. SDN networks have several benefits over conventional ones including easy management of networks and enforcing security policies. However, there are various reasons why SDN is usually not fully deployed in networks such as limited budget for new network infrastructure. In most cases, organizations do not usually like taking up huge budgets that would make them install new network infrastructure from scratch. Another reason is the fear of downtime that may be experienced during the transition to SDN. One way out of this could be deploying a few SDN-enabled devices together with the traditional or legacy network devices, thereby gradually replacing traditional network devices with SDN devices. A hybrid SDN network is one that consists of both SDN and legacy network devices.

Silviu-Gabriel Topoloi et al. [20] observe that despite SDN being criticized for some challenges like horizontal elasticity and real-time response, it seems that they have been solved and it is no longer a concept of academics alone but a success in business. For instance, *Google*, according to its own article [21], adopted SDN to connect its global data centers, which resulted into better operational efficiency reducing costs. Furthermore, NFV decouples network functions from specialized hardware thereby providing easy scaling and portability in conventional networking models.

According to *Diego Kreutz* [18], the standardization landscape for SDN includes organizations such as the Open Networking Foundation (ONF), the Internet Engineering Task Force (IETF), the International Telecommunication Union (ITU-T), and the Broadband Forum (BBF).

2.3.1 Network Function Virtualization

Rashid Mijumbi et al. [22] consider that traditional physical network equipment and rigid service chaining are major challenges faced by the telecommunication industry since they lead to low service agility, high costs and dependency on special hardware. In addition to that, is increasing customer demands for different services which forces Telecommunications Service Providers TSPs to continuously invest in new physical equipment. In this context, Network Function Virtualization (NFV) has been put forward as a possible answer to separate the function of a network equipment from the respective hardware.

The Network Functions Virtualization (NFV) is defined, by *Silviu-Gabriel Topolo et al.* [20], as a technology that separates the hardware that runs network functions (NFs) from the functions themselves, such as firewalls and routers, allowing NFs to be implemented as software programs that run on industry standard servers. This makes it possible to put together numerous network equipment types in high volume servers, which can be found both in data centres, distributed network nodes as well as end user premises. NFV introduces differences in network service provisioning, such as decoupling software from hardware, enabling independent evolution and maintenance for both software and hardware. This decoupling helps development timeline and, also, an independent evolution for both software and hardware, which aids adaptability and elasticity to legacy networking structures.

A Virtual Network Function VNF is considered, by *Rashid Mijumbi et al.* [22], to be known as a virtualized NF, such as a virtualized routers and firewalls, deployed on resources such as a virtual machine (VM). A service provided by a TSP consists of one or more NFs that can be virtualized and deployed on VMs when it comes to NFV. The performance of services, regardless of whether they are based on dedicated hardware or VMs, should look the same from the customer's point of view. In addition, the functioning and behavior specifications of VNFs that make up the service also determine how it will behave.

ETSI's NFV Management and Orchestration (NFV MANO) [23] seeks to achieve the necessary support for provisioning VNFs and their management operations like VNF configuration and infrastructure life cycle management.

Table 2: Comparison of SDN and NFV concept, Source: [22]

Issue	NFV (Telecom Networks)	SDN
Approach	Service/Function Abstraction	Networking Abstraction
Advantage	Promises to bring flexibility and cost reduction	Promises to bring unified programmable control and open interfaces
Protocol	Multiple control protocols (e.g. SNMP, NETCONF)	Multiple control protocols (e.g. OpenFlow, NETCONF)
Applications run	Commodity, servers and switches	Commodity servers for control plane and possibility for specialized hardware for data plane

Rashid Mijumbi et al. [22] discuss the relationship between SDN and NFV. They appear to have a lot in common, specially because both share some interests such as open source software, standard agnostic and vendor agnostic network hardware. To be more precise, NFV aims at running NFs on industry standard hardware and SDN controller can also be implemented as a software running on a industry standard hardware. In addition, both seek to leverage automation and virtualization to achieve their respective goals.

However, SDN and NFV are different concepts, aimed at addressing different aspects of a software-driven networking solution. NFV aims at decoupling NFs from specialized hardware elements while SDN focuses on separating the handling of packets and connections from overall network control.

Finally, both SDN and NFV can complement each other, for example, SDN controller may control and manage applications (such as load balancing, monitoring, traffic analysis, etc.) while VNFs benefit from NFV's reliability and elasticity features, and also, SDN can help accelerate NFV deployment.

Apart from that, Shuo Wang et al. [24] gives an example on possible uses for NFV and SDN in mobile edge networks: "A mobile network architecture that deploys and utilizes flexible computing and storage resources at the mobile network edge, including the radio access network, edge routers, gateways and mobile devices etc., with the help of SDN and NFV technologies".

2.3.2 Software Defined Networking Controllers

Between the variety of open-source solutions that exist for SDN Controllers, the chosen one will depend on the required characteristics for each goal. A research by Jhon Loza Lluco [25] shows that the industry has leveraged projects like OpenDayLight [26], created by the Linux Foundation in 2013, in collaboration

with vendors such as Brocade, Juniper, Cisco, Ericsson, IBM, Microsoft, NEC, Red Hat and VMware. OpenDayLight (ODL) is an open platform to configure and automate networks of any size and scale. There is also Open Network Operating System (ONOS) [27], which is a project from the Linux Foundation since 2014 that manages network components such as routers and links which also executes programs or software modules in order to create communication services to the final hosts and to a neighbor network. Finally, there is TeraflowSDN (TFS) [28] which is an ETSI project that has an SDN controller cloud-native for IP and optical networks with the help of big companies like Telefónica, NEC, CTTC, Atos, etc. In the end, all three options serve the same final purpose of configuring and automating networks.

ODL and ONOS are pretty similar, both were developed using the same architecture and a REST API for communication between modules and applications, and were both built on top of the same platform (Apache Karaf OSGi). The biggest difference between them is that ODL focuses on more various technologies and is oriented toward the new network generation. While ONOS focuses on performance and clustering on networks, growing their availability and scalability.

According to *Vaishnavi Moorthy et al.* [29], the flexibility to operate across multi-cluster environments makes ONOS useful for Service Providers. Its strong Web UI is more frequently a reason for selection as it makes networks easier to visualize and manage in comparison to ODL. In contrast, ODL was intended to be a general purpose controller designed for both longevity and flexibility supporting many different sorts of networks that use various kinds of network services, and thus perfect for addressing diverse use cases. After an evaluation on both controllers with respect to the performance metrics and topologies, ONOS seemed to be superior regarding packet forwarding when compared to ODL, this is easily explained by the ability to visualize the network traffic and a maintaining a high transmission rate of data packets.

Because of the similarities between OpenDayLight and ONOS, the table 3 below, shows the differences between OpenDayLight and TeraFlowSDN.

Table 3: Comparison between TeraFlowSDN and OpenDayLight, Source: [25]

	TeraFlowSDN	OpenDayLight
Deployment	MicroK8s, Docker	Apache Karaf OSGi, Docker
Requirements	4vCPUs, 8GB RAM, 60GB HDD, 1G NIC	2vCPUs, 2GB RAM, 16GB HDD
UI	GUI, CLI, GRAFANA	GUI, CLI
API	TAPI, SBI, NBI	API REST, SBI, NBI
Protocols	NETCONF, Openconfig, P4, gNMI...	NETCONF, Openconfig, Openflow, BGP...
Services	L2/L3 VPN/Slices, ACLs, IPv4...	BGP-LS, SR-TE, MPLS, L2/L3 VPN...
Integration	OpenSource MANO	Openstack, ONAP

TeraFlowSDN, as *Omer Bulakci* [30] points out, aims to deliver a new generation open-source cloud-native Software-Defined Networking (SDN) controller to provide smart connectivity services to beyond 5G networks. This project has proposed a new transport SDN architecture that, as *Ricard Vilalta* [31] explains, enables an open environment for network applications and devices using full standard interfaces with container-based services, which are deployed as micro-services and managed on elastic infrastructure through agile DevOps [32] processes and continuous delivery workflows.

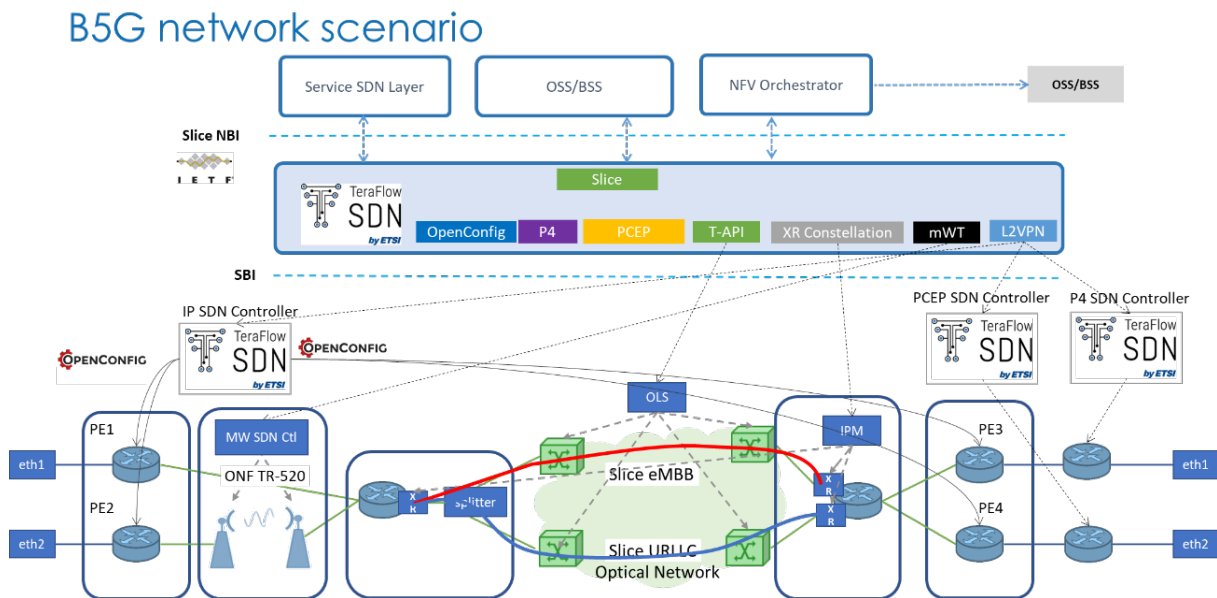


Figure 1: TeraflowSDN Scenario, Source: [33]

Figure 1 shows a scenario of *TeraFlowSDN* (TFS) providing an overview of its architecture, the types of infrastructure it communicates with, and the technologies that can orchestrate its network. TFS is an open-source, micro-service based, cloud-native and carrier-grade SDN controller, capable of integrating

with current NFV frameworks, within the ETSI community. This SDN controller is composed by multiple components which are treated as containers using virtualization techniques. These components, as *Omer Bulakci* [30] describes, are deployed as micro-services and managed on elastic infrastructure through agile DevOps processes and continuous delivery workflows.

In a micro-services architecture, services are simple and detailed, and the protocols are lightweight. This in particular, aims to control a network by bringing numerous advantages such as automation, monitoring, flexibility and scalability.

2.4 Quantum Key Distribution - Software Defined Networking

According to *Hua Wang et al.* [34], because of the fast development of Internet of Things (IoT) technology, a more secure communication for users is required with increasing demands in information networks, so as to overcome perceived security threats as much as possible. As a promising technology, quantum key distribution (QKD) has been proven, by *Charles H. Bennett et al.* [35], to provide users with secure keys exploiting the laws of quantum physics, i.e., Heisenberg's uncertainty principle [15] and the no cloning theorem [16].

QKD networks can be controlled by SDN for the unified interaction of network devices and protocols, as demonstrated by *Yuan Cao et al.* [36]. Similarly, *Yuan Cao et al.* in another article [37] suggest that QKD can be a secure solution for SDN-based networks.

It has been observed by *R. S. Tessinari et al.* [38] that it is becoming increasingly important to consider such quantum-safe communication solutions as advances in quantum computing edge closer to the prospect of eavesdroppers having sufficient computational power to break the computational hardness assumptions of classical public key cryptography.

The SDN approach is an important part of this, introducing clean abstraction of the control and data planes. This brings significant benefits, including more efficient resource utilization and simplified centralized management. Initial QKD SDN studies focused on customized implementations of interfaces, such as custom RESTful APIs or extending the OpenFlow SDN protocol. Since then, various standards have been developed for QKD, including a RESTful API for key delivery (ETSI GS QKD 014 [39]) and YANG models for both QKD Node (ETSI GS QKD 015 [40]) and QKD Orchestrator (ETSI GS QKD 018 [41]).

2.4.1 Architecture

The article written by *Alejandro Aguado et al.* [42], talks about the deployment of QKD-devices on an SDN network, and presents a quantum sensing SDN architecture which divides the network into three layers: application layer, control layer, and lastly, infrastructure layer. This SDN architecture is designed to address the challenges of merging quantum and classical networks, such as managing quantum channels and reducing noise interference. The authors also propose an end-to-end encryption for secure network management and highlight potential cost savings. In terms of orchestration, a distributed NFV architecture is suggested by combining NFV orchestration with QKD technology.

Similarly, *Wanrong Yu et al.* [43] uses a three-layered architecture as well, but focus on researching some possible changes in order to solve problems, related to the management complexity of QKD networks, such as excessive key consumption and inability to achieve random routing. Their solution shows an improvement related to secure key consumption and also a better availability and performance of the QKD network.

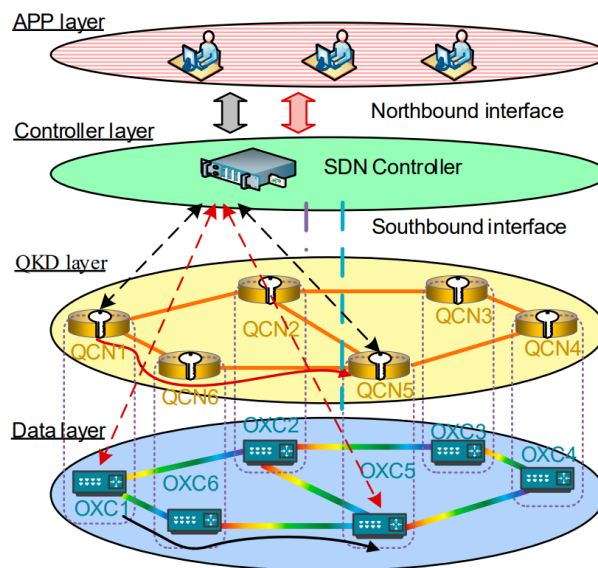


Figure 2: QKD network architecture, Source: [34]

In addition, as shown in Figure 2, a four layered architecture was proposed, by *Yongli Zhao et al.* [44][34], in order to integrate QKD into existing optical networks. The difference between the three-layered and the four-layered architecture is just a split of the whole infrastructure layer into a QKD layer, for the quantum channels, and a data layer, for the optical channels. Below is a description of each layer:

- **App Layer:** An application is, considered by ETSI [40][41], an entity consuming QKD keys from the key management system. They can be either external applications or internal applications running

in the QKD system.

Hua Wang et al. [34] states that the application layer is present at the top of the architecture of the QKD networks enabled by SDN. It directly faces demands of users and also abstracts network resources for users. Unlike classical optical networks, the application layer includes two major services, i.e., secret-key provisioning services and security management services. Secret-key provisioning services provide secret keys for the security demands of the networks such as authentication, encryption, and signature. Security management is mainly responsible for the functions such as intrusion detection, virus protection, and security posture sensing, etc.

The QKD controller does not deal with keys itself. Keys are supplied directly from a Key Manager to a cryptographic application (as per ITU-T specifications [45]). In this case, once the keys are supplied to the cryptographic applications, they use the keys under their own responsibility (following ITU-T recommendations [46]).

- **Control Layer:**

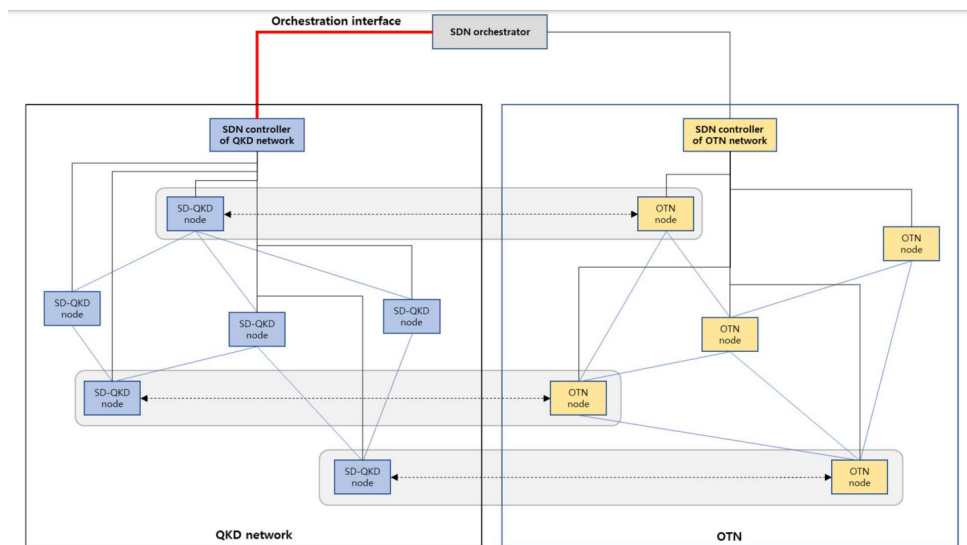


Figure 3: Use case of SDN orchestrator and SDN controllers for QKD network and Optical Network, Source: ETSI GS QKD 018 [41]

A QKD controller is, considered by ITU-T [45][46], a functional module, which is located in a quantum key distribution (QKD) network's control layer to control a QKD network.

As noted by ETSI [41], Control layer provides a holistic view of QKD networks for the administrator. This layer may include one or multiple controllers to implement network management over QKD layer and open network capabilities for various applications. In case of Figure 3, there are two SDN

controllers, one specific to the QKD network and the other to a classical Optical network. Also, an SDN orchestrator might be present to orchestrate multiple network domains via each SDN controller of each network as well as both QKD and classical network domains. It enables the provisioning of end to end services through different network domains. Different numbers of controllers in control layer, as *Hua Wang et al.* [34] pointed out, can support hierarchical structure and multiple domains to improve the scalability of the networks.

Specifically, application layer receives demands from operators then generates requests and sends them to the controller through its northbound interface (ETSI, [41]). Similarly, there is a connection between the controller and each node through its southbound interface (ETSI, [40]). From this last interface, the controller is able to create a topology view and allocates QKD resources. Correspondingly, control layer controls the QKD resources in QKD layer, provides services for multiple applications in application layer, and receives resource allocation and policy information of the key distribution layer, as discussed by *Hua Wang et al.* [34].

- **QKD Layer:**

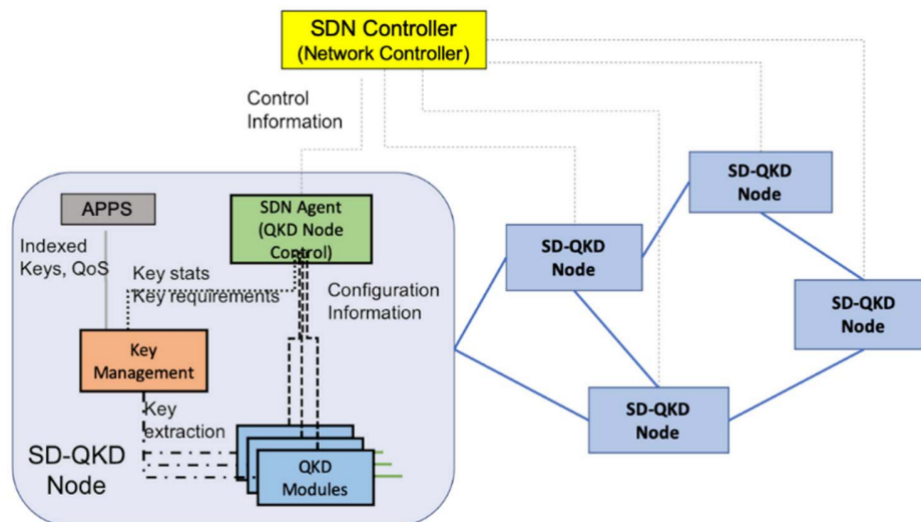


Figure 4: Depiction of an QKD SDN network showing a set of QKD nodes, Source: ETSI GS QKD 015 [40]

This new layer, which is not present in any current SDN architecture, corresponds to the QKD nodes and the communication between them which is mostly used for key exchange.

ETSI [40] states that a QKD node is an aggregation of one or multiple QKD Modules that interface with an SDN controller using standard protocols. Figure 4, depicts a set of QKD nodes interconnected under the control of the SDN controller. One of the nodes is shown in more detail with the fundamental components. This SDN node is composed by:

- SDN Agent, which acts as a small controller for the node itself where it configures and controls every QKD Module inside the node as well as the Key Management.
 - QKD Modules, which exchange keys with other QKD Modules using a direct or a multi-hop connection with the help of key relaying for the latter (*ITU-T Y.3803* [46] contains three different possibilities to achieve this exchange for a multi-hop connection).
 - Key Management, which extracts the keys from the QKD Modules and stores it in its database for managing and providing it to the respective Apps.
 - Apps component, which is also present, although, this was only included as part of the node to illustrate that they are contained in the same security perimeter.
- **Data Layer:** This last layer isn't related to QKD SDN but, for a context, corresponds to the optical layer for actual data transport.

2.4.2 Interfaces

ETSI [40][41] recommends the use of YANG [47] for data modelling because it is considered the main modelling language for network elements, systems and services while the most of the others network control planes protocols already use it as well. Apart from that, YANG is also easy to define, read and extend. Both RESTCONF [48] and NETCONF [49] protocols use YANG.

Underneath follows an explanation, by ETSI [40], on both most used protocols for SDN:

- RESTCONF [48]: usually used in the SDN controller and other platforms' interfaces, it is based on HTTP in order to implement CRUD operations (create, read, update and delete) over data defined by YANG. Any call to this interface requires, among other parameters or headers, an URL and a body (usually encoded in JSON or XML).
- NETCONF [49]: in this case, it is usually implemented between a network element and an SDN controller or Network Management System (NMS). Its encoding is usually XML (it also accepts JSON), and the protocol is based on transactions, while the transport is usually over secure protocols, such as SSH or TLS.

On the SDN controller view, there is a SouthBound Interface (SBI) to the below infrastructure and a NorthBound Interface (NBI) connecting to the SDN orchestrator, each interface is described by both ETSI's standards [41][40], respectively, as follows:

- **NorthBound Interface (NBI):** As shown in Figure 3, there is a connection between the SDN orchestrator and the SDN controller. Northbound interface is used for the SDN orchestrator to control all different network domains through each SDN controller, allowing an automatic coordination of the available resources and an optimal path for an end-to-end service between different network domains. *Hua Wang et al.* [34] states that the basic control functions of it include functions such as topology acquisition, service request, link building and path calculation.
- **SouthBound Interface (SBI):** As shown in Figure 4, there is a connection between the SDN controller and the SDN agent which uses this SBI. Southbound interface is used for the SDN controller to control the network in terms of routing, configuration, policy-based, access and session controls. Each type of control can be found on *ITU-T Y.3804* [45]. For monitoring purposes, each infrastructure element (each QKD node in a QKD network) should notify the SDN controller about their current health and status from time to time or when any change happens.

2.5 Final remarks

Since Quantum computers aren't yet on market there aren't related works about this specific topic (QKD SDN). Until today the most related work that exist are currently the specifications from ETSI [40][41] and from ITU-T [46][45].

In the next chapter, TFS will be described into more details such as architecture and functionality. A similar description will be done about QKD network as well. Apart from it, it will also be presented a possible integration of the QKD network into TFS. Lastly, an example on how to use TFS with this new integration and the respective test cases will be shown.

Chapter 3

Concept and Design of QKD into TeraflowSDN

In this chapter, the fundamental architectural concepts of TeraflowSDN and the proposed QKD Network are presented. It enumerates all components from TeraflowSDN which are most important to this project and, then, it introduces concepts about each component of the QKD Network.

3.1 Advantages of TeraflowSDN

In the context of development it was opted to use the vendor-agnostic SDN Controller mentioned above known as TeraflowSDN (TFS) for reasons such as:

- Scalability – Handles large, complex networks with ease.
- Dynamic Flow Management – Offers real-time, flexible traffic handling.
- Performance Optimization – Low-latency routing for improved network efficiency.
- Service-Aware Networking – Optimizes services to meet performance requirements.
- Multi-Vendor Interoperability – Works with existing multi-vendor infrastructures.
- Security – Provides robust security features like flow-based access control.
- Automation & Orchestration – Automates complex network tasks for efficiency.
- Customization & Flexibility – Highly customizable to fit specific needs.
- Open-Source – Offers transparency and community support.
- Support for Emerging Technologies – IoT and edge computing.
- Cost Efficiency – Potentially more affordable than proprietary SDN solutions.

3.2 TeraflowSDN Architecture

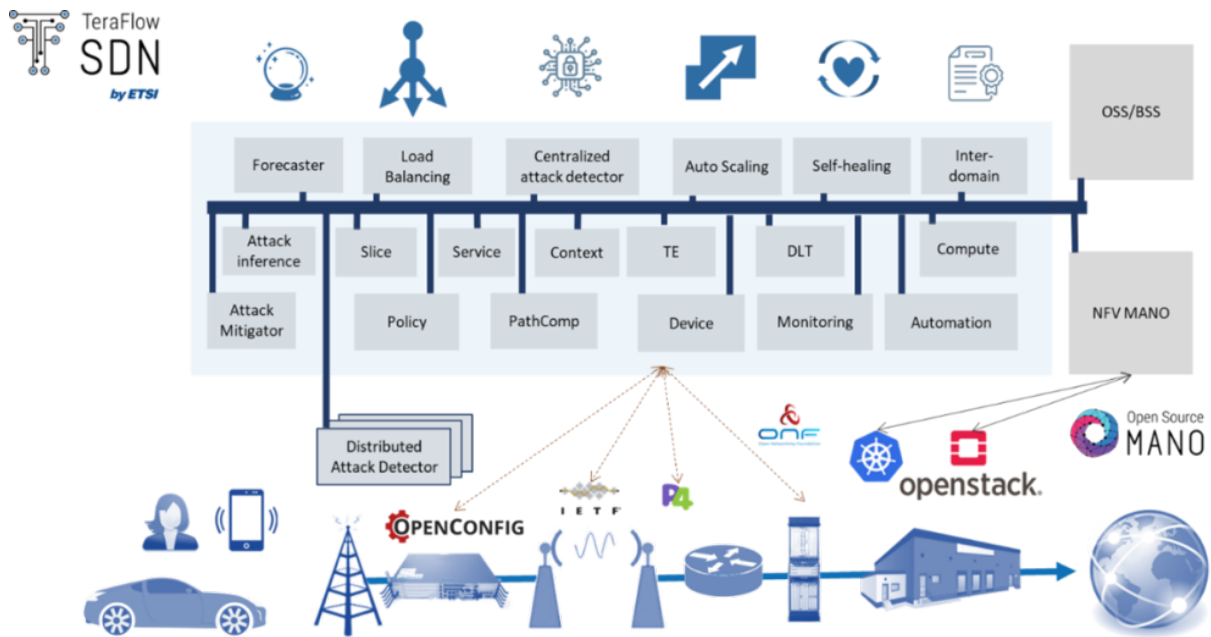


Figure 5: TeraflowSDN Architecture, Source: [50]

Figure 5, presents an architecture of the current *TeraflowSDN*. The architecture includes several components, as shown in the figure. A subset of the most relevant ones for the integration is described below. Each component acts simultaneously as a Remote Procedure Call (RPC) [51] client and server allowing each component to communicate with the other ones.

- **WebUI** → *TeraflowSDN*'s Web User Interface. After the user provides a descriptor file containing at least the topology, the webui shows a graph of the topology itself with each device and links. Apart from the topology it also makes it possible for the user to add new devices, create new services/slices. When it comes to monitoring, *TeraflowSDN* also provides a bunch of dashboards with the help of *Graphana* to visualize the collected metrics.
- **InterDomain** → Enable interaction between different instances of *TeraflowSDN*. This allows different networks controlled by different SDNs to communicate between each other.
- **DLT** → Provides interface with blockchain for the Gateway inter-domains communication. It uses *Hyperledger Fabric* framework to accomplish its needs.
- **Device** → It's considered the SBI of the SDN controller. In charge of interacting with the different

network and infrastructure elements (Optical Switches, etc.) through their respective interfaces such as P4, OpenConfig, etc. (More examples in Figure 1)

- **Context** → Considered the core of the whole *TeraflowSDN*. It contains a database where it keeps all information about the topology, devices and their configurations, created services, etc.
- **PathComp** → When a service is requested, the *SDN* has to choose an optimal path for it. This component computes the best path for both intra-domain and inter-domain services applying different algorithms (Shortest path by default) and taking into account any specified restrictions or limits.
- **Service** → In charge of creating and managing connectivity services. After receiving the service request it asks *PathComp*'s component to compute an optimal path between the two endpoints based on the specified restrictions/limits. Finally, having the desired optimal path, this component requests the *Device*'s component to communicate with each device in the path providing their respective configuration rules.
- **Slice** → In charge of creating and managing the transport network slices which appeared with 5G.
- **Monitoring** → In charge of collecting monitoring data from the different elements in the infrastructure. Uses *Prometheus* to collect the metrics provided by each component and, finally, displays the dashboards with the help of *Grafana*.
- **Compute** → Exposes an NBI API towards external systems such as an Open Source MANO or other NFV or MEC frameworks for interaction with the *TeraflowSDN* Controller.

3.3 How QKD Network works

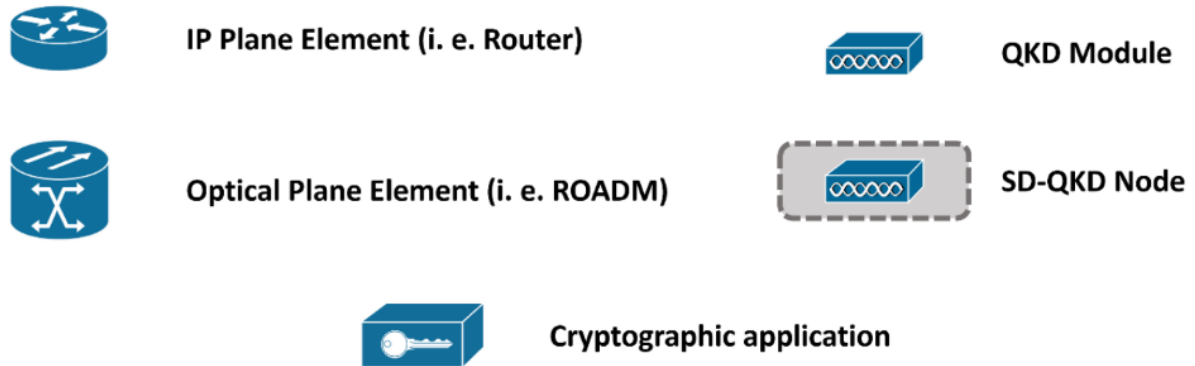


Figure 6: Network Elements, Source: [52]

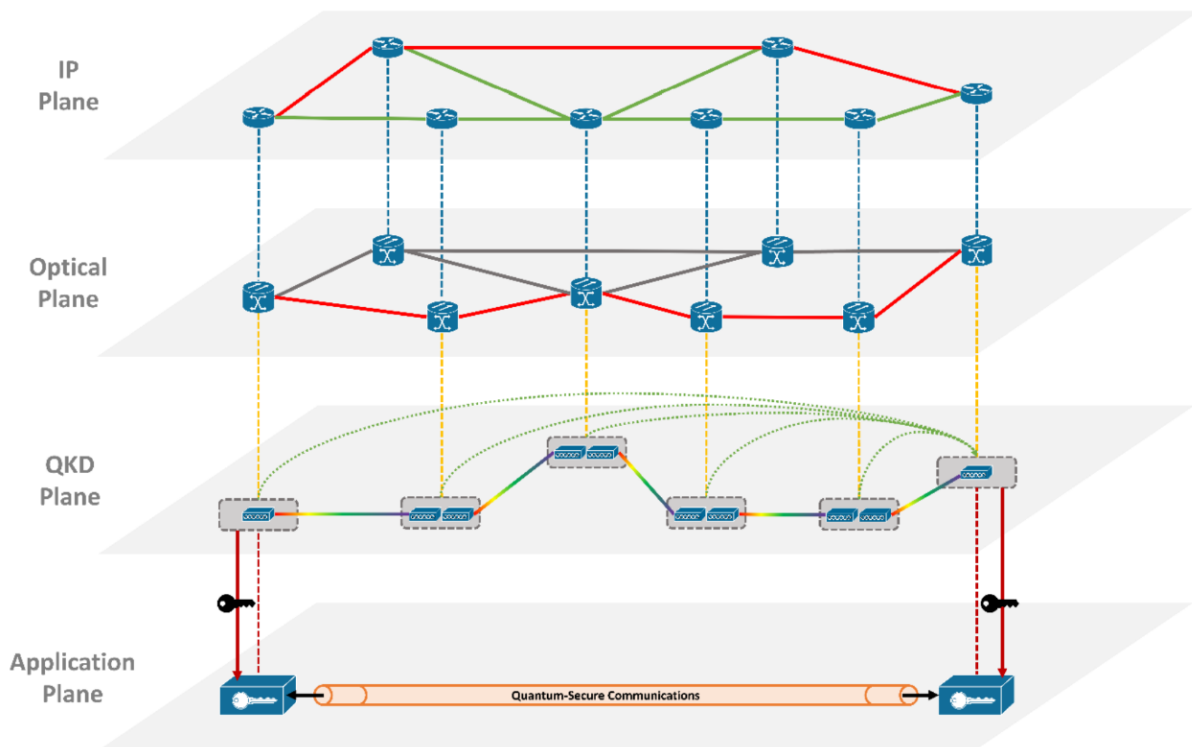


Figure 7: Network Provision for Quantum-Secure Communications, Source: [52]

Phase 1: Pre-Provisioning of QKD Links

In an initial phase the QKD Links between QKD Nodes are provisioned. At this stage, the topology of Optical, IP and QKD are already known, where their correspondences are represented by dotted lines. However, there is no key relaying process established nor an application plane.

Phase 2: Discovering Correspondence between Cryptographic Applications and Network Elements

In the second phase, the correspondence between cryptographic applications (present in the application plane) and the networks elements (more specifically, the QKD elements) is established.

Phase 3: QKD Plane Configuration and Key Relay Calculation

In the third phase, for a possible QKD Key exchange between the two matched QKD Nodes from the previous phase, the QKD plane is configured and the Key Relay process is calculated, which is represented by the dotted lines connecting all the QKD Nodes to the last QKD Node. This configuration ensures that QKD Keys can be securely relayed between the QKD Nodes in the QKD plane.

Phase 4: Provisioning of IP Resources

In the fourth phase, the IP SDN Controller prepares the IP Network (in the IP plane) to be able to handle the communication between the two Applications. The links in the IP plane allocated for this communication are colored in red.

Phase 5: Application Plane Configuration and Registration

In the last phase, the application plane has already a connection both in the IP Plane for the communication and a connection in the QKD plane for the key exchange. The applications are now ready to request keys from the KMS and establish quantum-secure communications.

3.4 QKD network Architecture

This section aims to explain the definition and a detailed description of the current QKD network architecture based on the contents provided by the project Opensec derivable E6 [52]. This description consists of every component in use, their interfaces and how they will communicate with each other. Figure 8 presents the proposed architecture in the aimed deployed environment. From the Figure it can be understood that there is an SDN Orchestrator in charge of orchestrating every SDN Controller and an SDN Controller for each different type of network (Optical, IP and QKD).

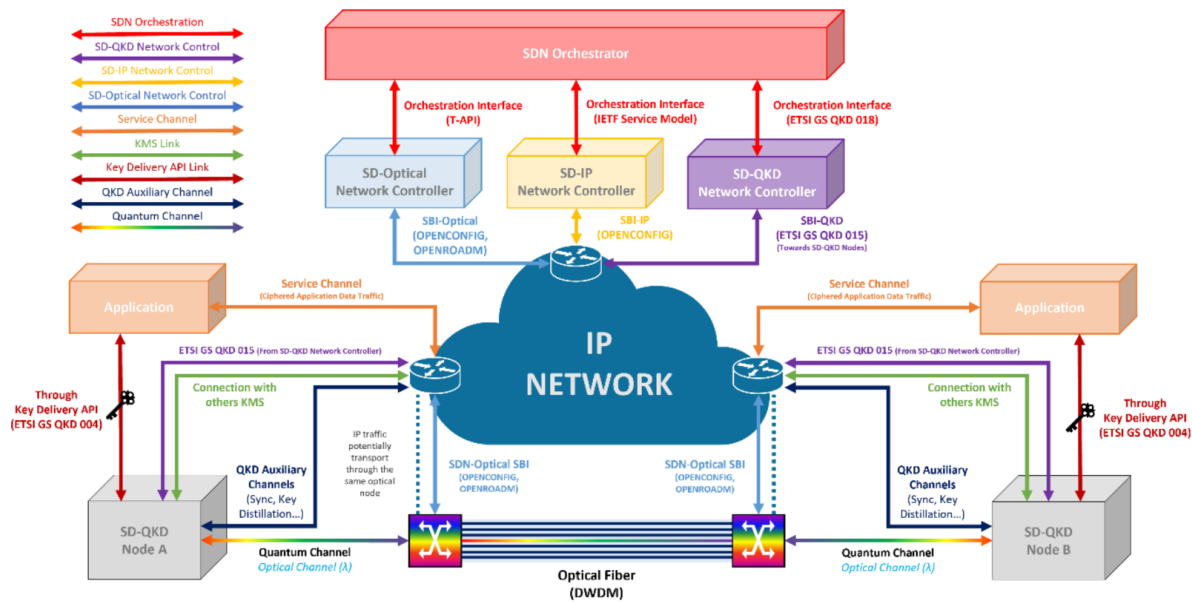


Figure 8: QKD Network Architecture, Source: [52]

For the QKD network to operate it requires at least two QKD Nodes which must be connected by a QKD Link to establish the communication between them. These QKD Links are composed by quantum channels in the optical layer and auxiliary QKD channels as showed in the figure, where each Quantum Channel is sent through an Optical Fiber (DWDM) along with classical channels, thus allowing the coexistence of quantum and classical channels.

Regarding the process of the actual propose of this architecture, to have QKD keys being distributed to each application it requires a previous configuration of each component present in the network. Initially, the optical plane must be adapted and configured to allocate the necessary resources for the quantum channels. Not only the Optical but also the IP plane requires a previous configuration as well. The application layer must be informed of the QKD endpoint to which it will request the QKD keys. In the end, to ensure everything is working, the QKD plane must be configured so a QKD link will connect both ends in order to allow the exchange of QKD keys between applications which is the main goal.

Apart from the components present in Figure 8, it is also possible to observe a few interfaces, more specifically, ETSI GS QKD 014 [39] related to the delivery of keys from the node to the respective application, ETSI GS QKD 015 [40] related to the communication between the QKD SDN and each QKD Node and, finally, ETSI GS QKD 015 [40] related to the orchestration of the QKD SDN.

3.4.1 QKD Node

A QKD Node contains all software and hardware to establish a QKD connection (QKD Key Association Link) between nodes which must be connected directly by a physical QKD Link or by means of multi-hopping to each other. After establishing a QKD Key Association Link between QKD Nodes, each node from the ends must also provide QKD keys to the applications requesting them. A QKD Node must be inside a secured area agreeing with the prescribed protocol and must also be unmodifiable and impenetrable.

A QKD Node is composed by the following elements:

- QKD Module (1 or more)
- Key Management System
- QKD Node Controller

Figure 9, shows the simplest QKD Node with the minimum requisites to be functional. All interfaces used are also present.

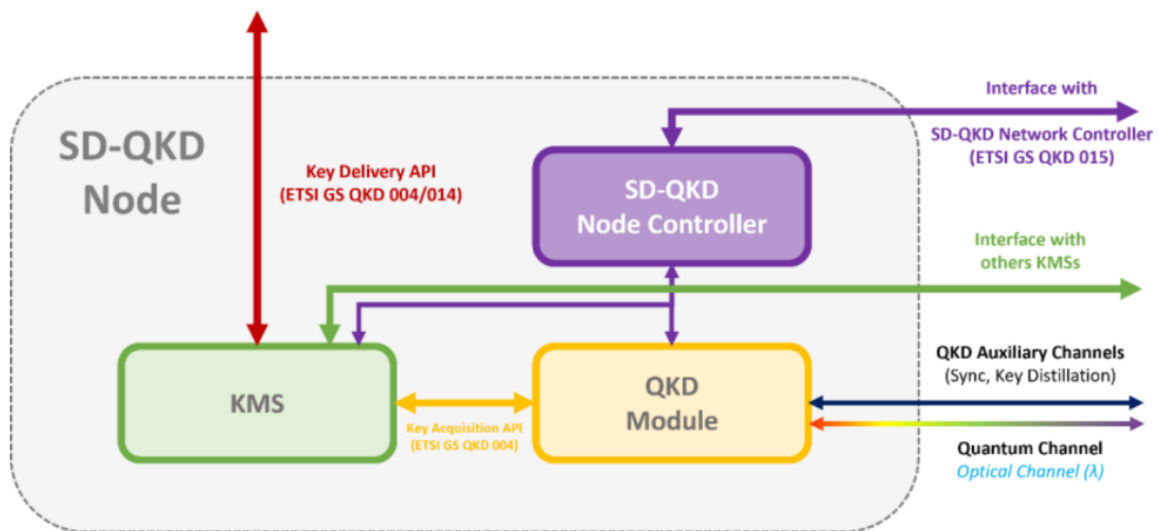


Figure 9: Minimum QKD Node, Source: [52]

More complex QKD Nodes contain multiple QKD Modules, specially to enable multi-hop key relay, and that also comes at the cost of a bigger security attention. Figure 10, presents a QKD Node containing three QKD Modules. These QKD Nodes which enable multi-hop key relay, known as Trusted Nodes, play an important role in the current QKD Networks.

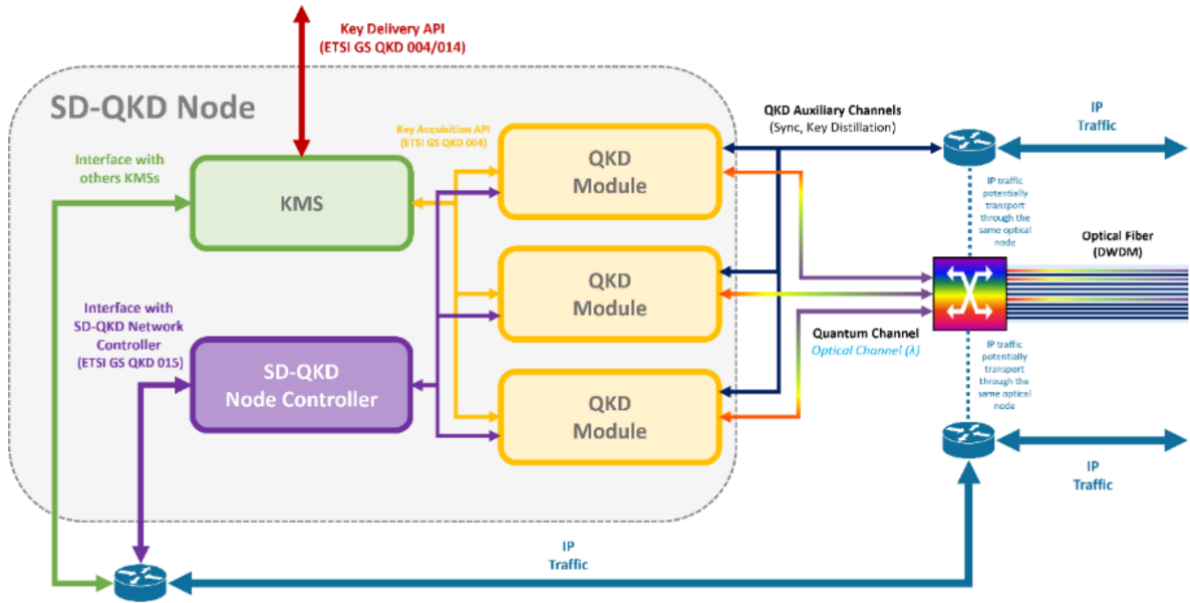


Figure 10: Complex QKD Node, Source: [52]

QKD Module

A QKD Module, illustrated in Figure 11, is a group of hardware and software which carries out cryptographic algorithms and quantum optical processes such as QKD protocols, synchronization, distillation of the key generation and is implemented inside a cryptographic border. In Figure 11, not every component is required and it's also necessary to specify that vendors could have different implementations of this architecture. Since this work is not focused on the composition of the QKD Module, it will not provide further details about the architecture.

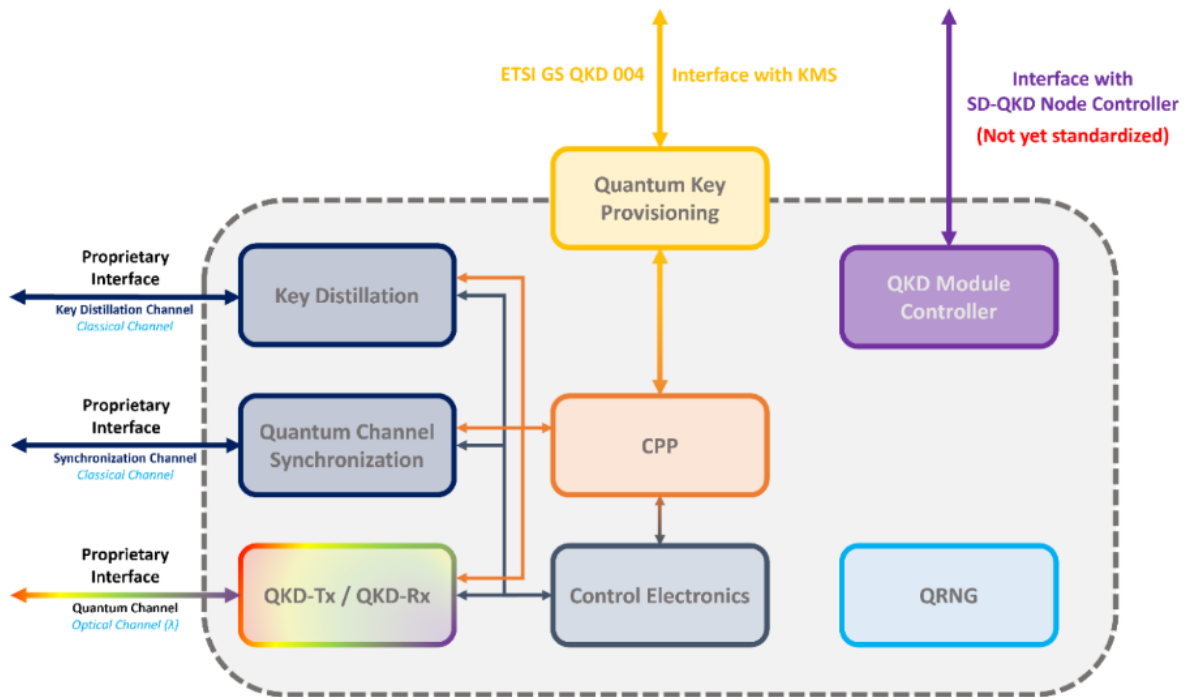


Figure 11: QKD Module, Source: [52]

Two QKD Modules generate a pair of symmetric (identical) random bit strings which is called a QKD key that follows the secure protocol of QKD. Those keys are then sent to the KMS to be stored and later shared with the applications if requested by them. The communication between the QKD Module and the KMS is supported by ETSI GS QKD 004 [53] standard.

The brain of the QKD Node, named QKD Node Controller, requires a communication to each QKD Module for configuration exchange, monitoring and execution of different control and management tasks on the QKD Modules.

Key Management System

The KMS is a component part of the QKD Node which manages and stores QKD Keys. Key management refers to all action applied on keys during their life cycle. Their life cycle initiates from the reception from the quantum layer up until their deletion or preservation depending on the key management policy. Throughout their life, a key can be stored, formatted, relayed, synchronized, authenticated and, most importantly, used to supply a cryptographic application.

A QKD Network contains multiple QKD Nodes connected by QKD Links. For a QKD Node to connect to another QKD Node without being directly connected through a QKD Link it needs to rely on trusted nodes, intermediate QKD Nodes that help extending the reachability and availability of key supply. These

Trusted Nodes can relay QKD Keys allowing a QKD key to travel between any two parties, even if they are not directly connected by a QKD link. ITU-T Y.3803 [46], presents multiple possibilities for key relays but, for this project, only one will be chosen, which is considered to be the most optimal and safest.

Beneath, it will be presented the proposes from ITU-Y.3803 [46] related to QKD Key relaying between QKD Nodes. In the cases below, the Key Management Agent (KMA) is responsible for storing and managing the life cycle of the keys in its trusted node and the Key Supply (KSA) is responsible for supplying those keys to applications or to other nodes.

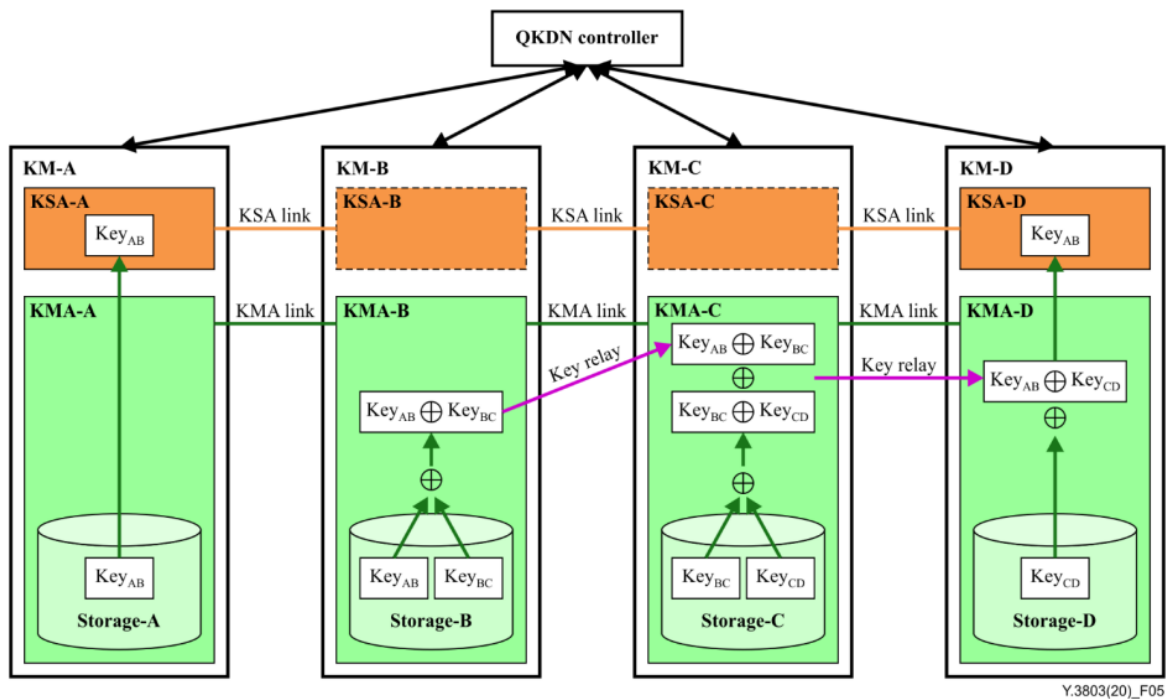


Figure 12: Scheme of key relay from point to point (Case 1), Source: ITU-T Y.3803 [46]

Figure 12, Case 1, presents a key relay scheme that allows the sharing of keys between source node and destination node through multiple trusted nodes. The Key_{AB} is generated between KMA-A and KMA-B. The Key_{AB} is relayed from KMA-B to KMA-C by OTP encryption with the Key_{BC}. It is relayed from KMA-C to KMA-D by OTP encryption with Key_{CD} and finally supplied to KSA-D.

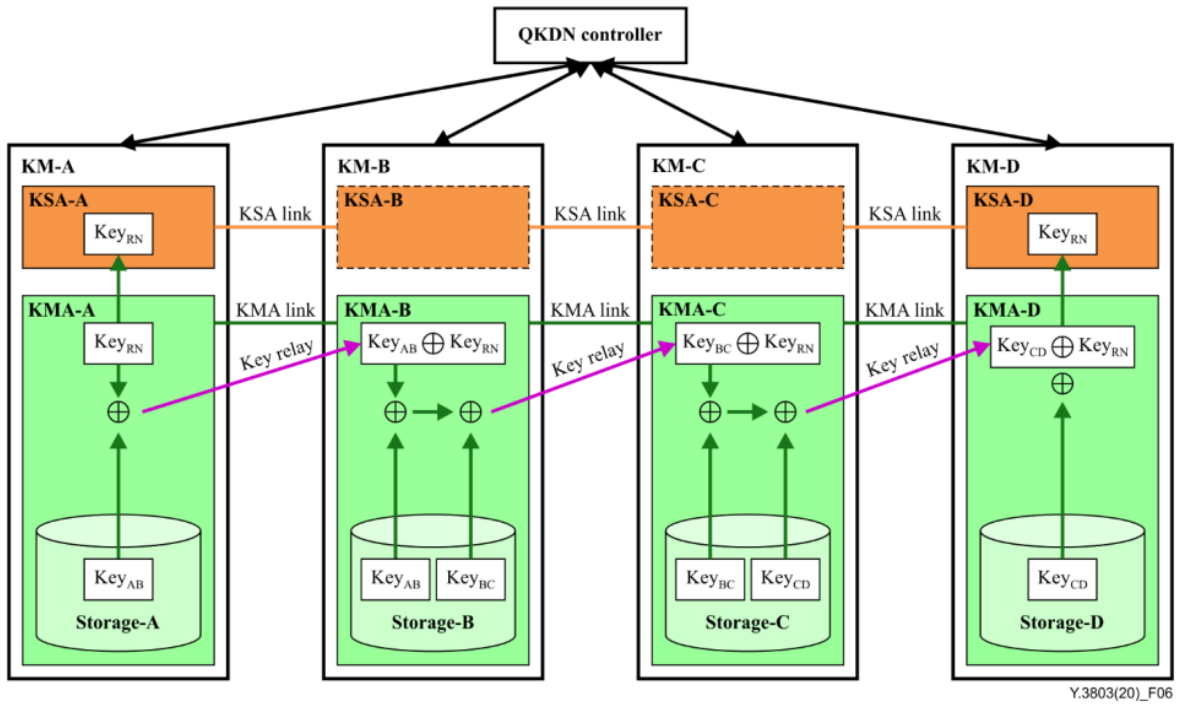


Figure 13: Scheme of key relay from point to point (Case 2), Source: ITU-T Y.3803 [46]

Case 2, represented by Figure 13, shows a random bit string KeyRN being generated locally at KMA-A and used for key relay from KMA-A to KMA-D.

Both Case 1 and Case 2 appear to face a problem when it comes key relaying from point to point in real applications due to all nodes ending up having knowledge about the XOR cipher text key and the respective decryption key, which is a security risk. Another problem is that the network has complex KMA links between QKD Nodes.

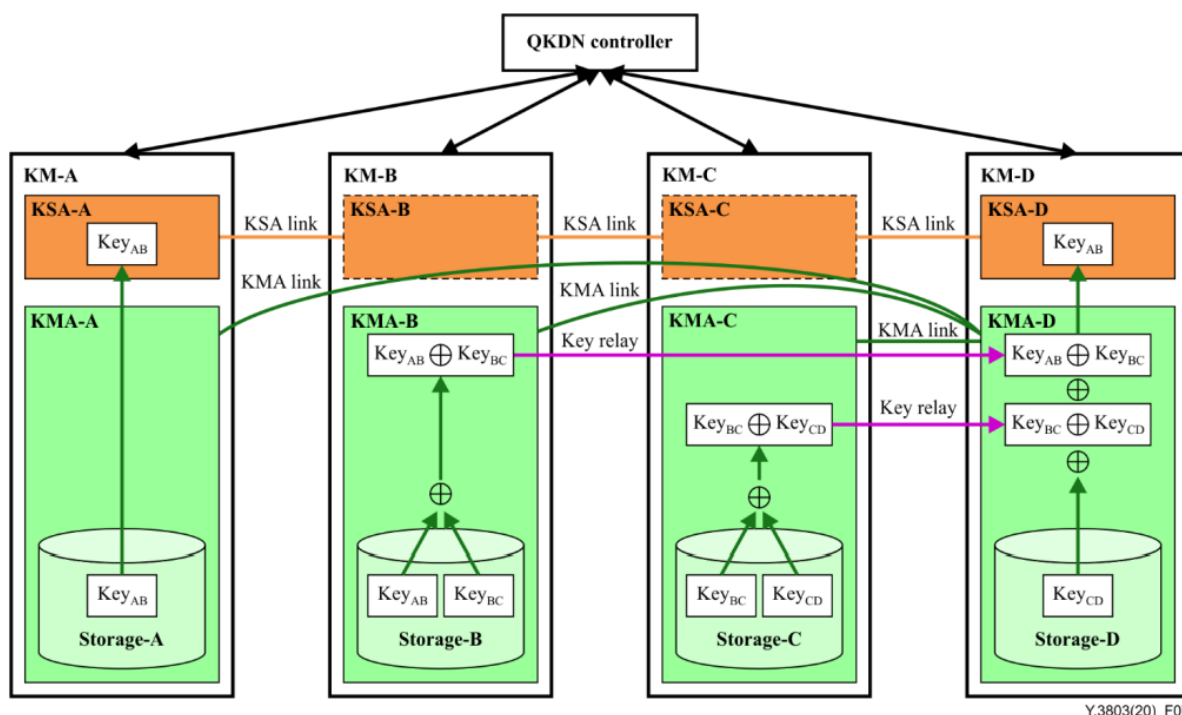


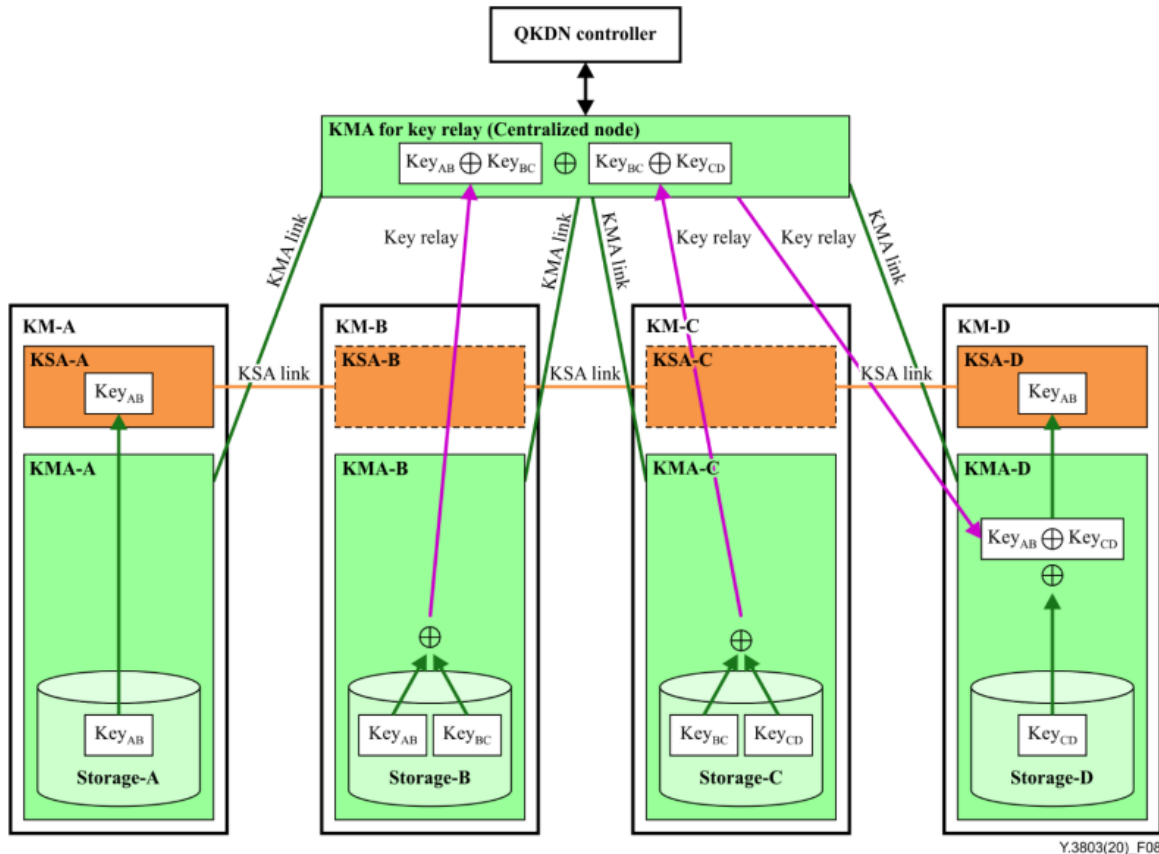
Figure 14: Scheme of key relay with XORs uniformly processed at destination node (Case 3), Source: ITU-T Y.3803 [46]

In order to help mitigating the problems faced by Case 1 and Case 2, Case 3 comes with the following solution, presented in Figure 14:

This scheme is similar to the scheme of Case 1 (12), apart from three different points:

- KMA link of each node connects directly to the destination node instead of each neighbour.
- The XOR ciphertext key is sent uniquely to the destination node.
- The relaying key is only decrypted at the destination node by XORing all the ciphertext received thus granting a higher level of security (no more knowledge of the key in each intermediary node).

This scheme from Case 3, simplifies the function of the key relay node. The XOR ciphertext key is only sent to the destination node and no neighbour node has to relay the key. This way, the key won't be shared with the intermediary nodes ever, making it more secure. However, complex KMA links is still an issue in the network with this approach. Because of that, it's only in use into some specific scenarios like the trunk-line network for link length extension.



Y.3803(20)_F08

Figure 15: Scheme of key relay with XORs collected at one centralized node (Case 4), Source: ITU-T Y.3803 [46]

Another scheme was proposed in contrast to the one presented in Case 3 (Figure 14) by applying the following changes:

- Computation of the XOR algorithms are all executed in a centralized node which will then send the final XOR ciphertext key directly to the destination node.
- Each node has a KMA link connected to the centralized KMA instead of another node.
- The QKDN (QKD Network) Controller is connected to the centralized KMA instead of being connected to each node.

This scheme not only simplifies KMA links since they are all connected to a centralized node, but also, facilitates network implementation.

From all this options available, for this project, it will be chosen the scheme based on Case 3 (Figure 14), because it is the most optimal and safest of all the three first cases and doesn't rely on a centralized node like the fourth case.

Below is an architecture of the KMS components along with its interfaces:

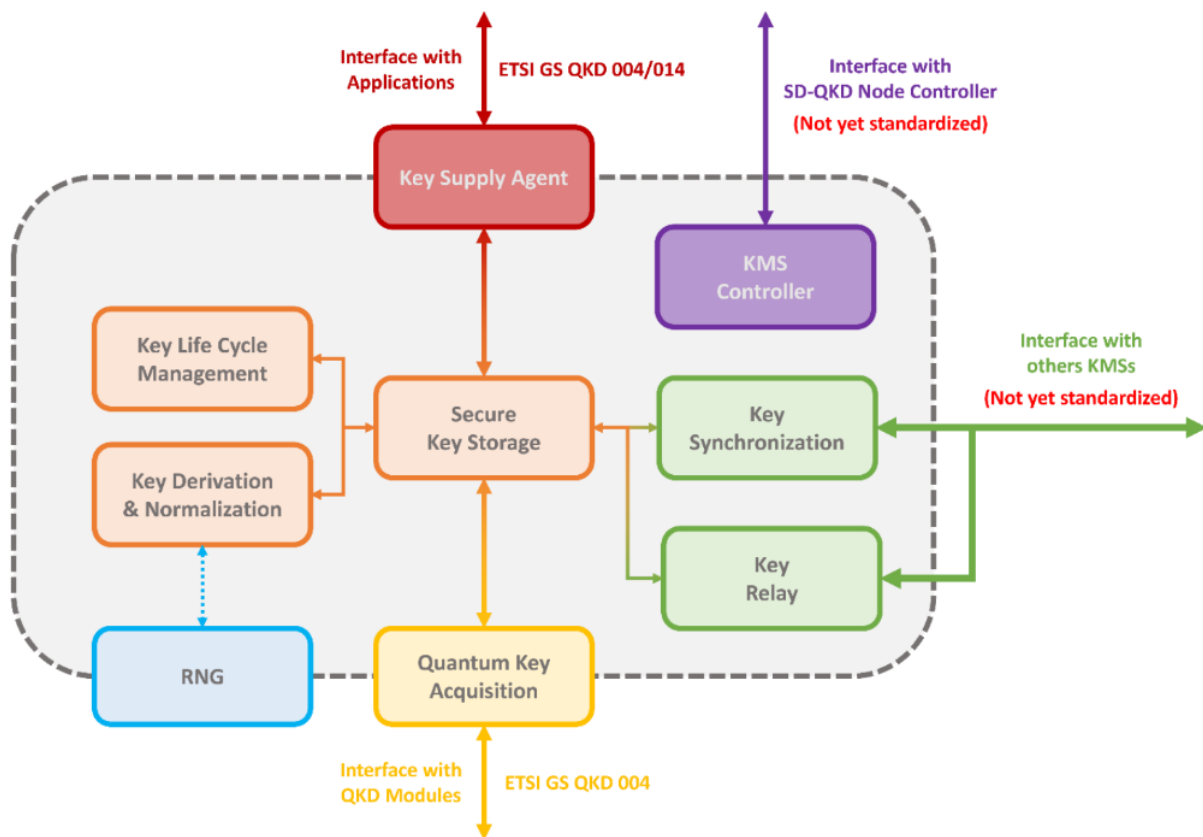


Figure 16: KMS (Key Management System), Source: [52]

Like the QKD Module, since this thesis isn't related to the composition of the KMS it won't go into any more details about the architecture.

The KMS has 4 different exterior connections which will be explained below:

- QKD Modules: Supported by ETSI GS QKD 004 [53] standard, the KMS has an interface connected to the QKD Modules in order to exchange the keys for management of their life cycle.
- Others KMSs: It's responsible for the key relay, synchronization and verification of their integrity.
- QKD Node Controller: KMS establishes communication with QKD Node Controller in order to exchange configurations and monitoring information.
- Applications: Supplies QKD keys to the respective applications requesting them. It is supported by ETSI GS QKD 004/014 [53][39] standard.

QKD Node Controller

The QKD Node Controller (also known as SDN Agent in ETSI GS QKD 015 [40] terminology) is responsible for the configuration, management and monitoring of all components present in the QKD Node. It gathers data of the overall status of the QKD Node along with information from the QKD Modules and KMS which is later sent to the QKD Network Controller. Since this thesis isn't related to the composition of the QKD Node Controller it won't go into any more details about the architecture.

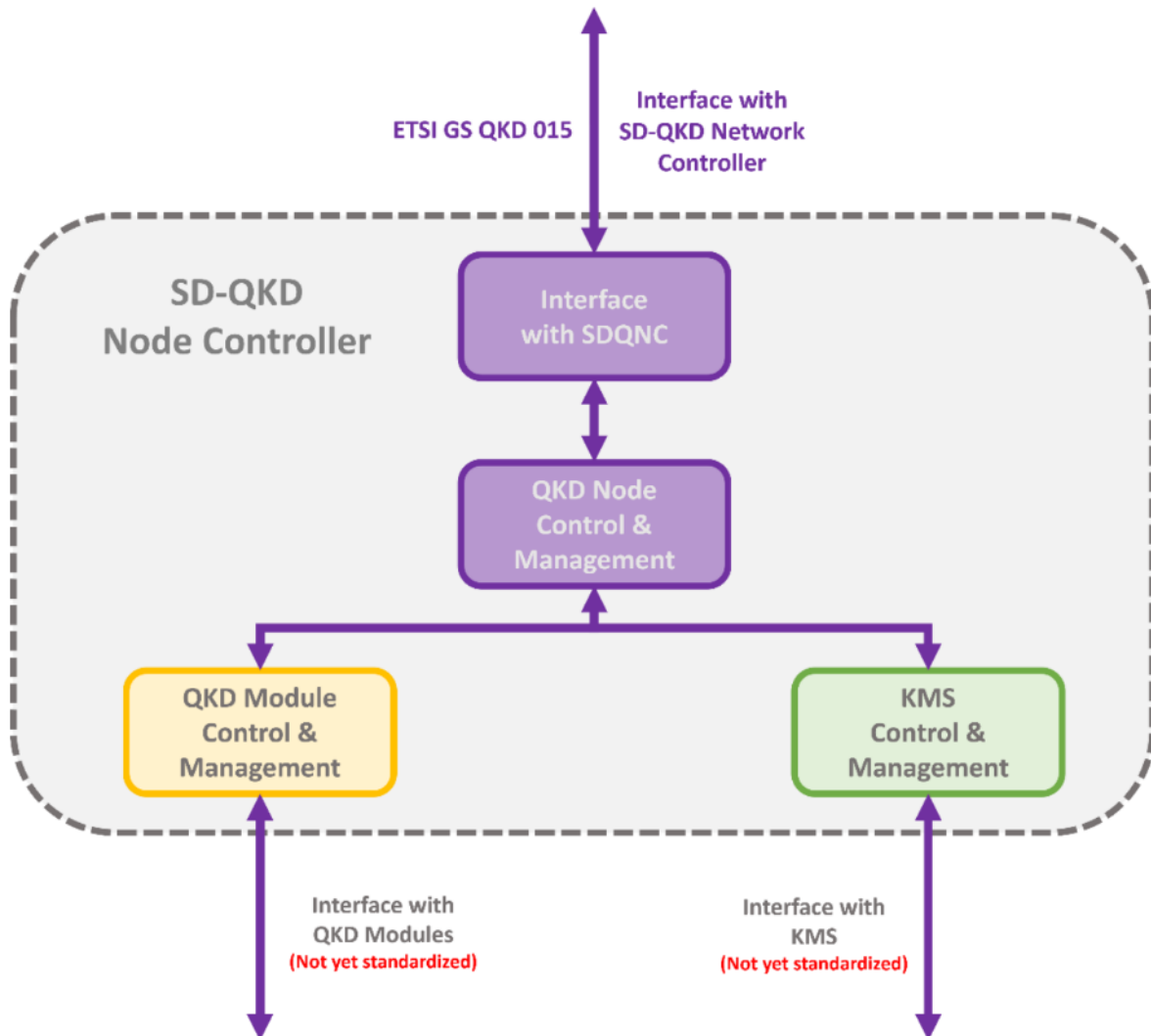


Figure 17: QKD Node Controller, Source: [52]

This component has the following three types of connections, each for a specific set of components present in the QKD Node:

- QKD Network Controller: Responsible for the communication with the QKD Network Controller. From this connection, the QKD Node Controller also receives orders of management and configu-

rations. This connection is supported by ETSI GS QKD 015 [40] standard either by a RESTCONF or NETCONF protocol both being based on YANG models.

- QKD Modules: It is also responsible for controlling and managing the QKD Modules by following the configurations provided by the QKD Network Controller or in case of a trigger, then the QKD Node Controller requests the change by itself. Apart from the configurations, the QKD Node Controller must gather the metrics from monitoring QKD Modules. The QKD Modules must send a set of monitoring parameters to the QKD Node periodically or when changes happened (depending on the configuration established).
- KMS: Similar to the connection between QKD Node Controller and QKD Modules, the connection between the QKD Node Controller and the KMS is essential for the QKD Node Controller to control and manage the KMS action based on the configuration provided by the QKD Network Controller or in changes detected by itself based on triggers. The QKD Node Controller decides which parameters are relevant for monitoring purposes from the KMS and receives that monitoring data periodically or on change such as new applications registering in the KMS to request QKD Keys.

3.4.2 QKD Network Controller

The QKD Network Controller is an SDN controller responsible for the configuration, management and monitoring of the QKD Nodes from a network by following the standard protocols and data models defined in ETSI GS QKD 015 [40]. It also exposes an API for an external system, also known as QKD Orchestrator, in order to orchestrate the entire network below by following the standard protocol defined in ETSI GS QKD 018 [41].

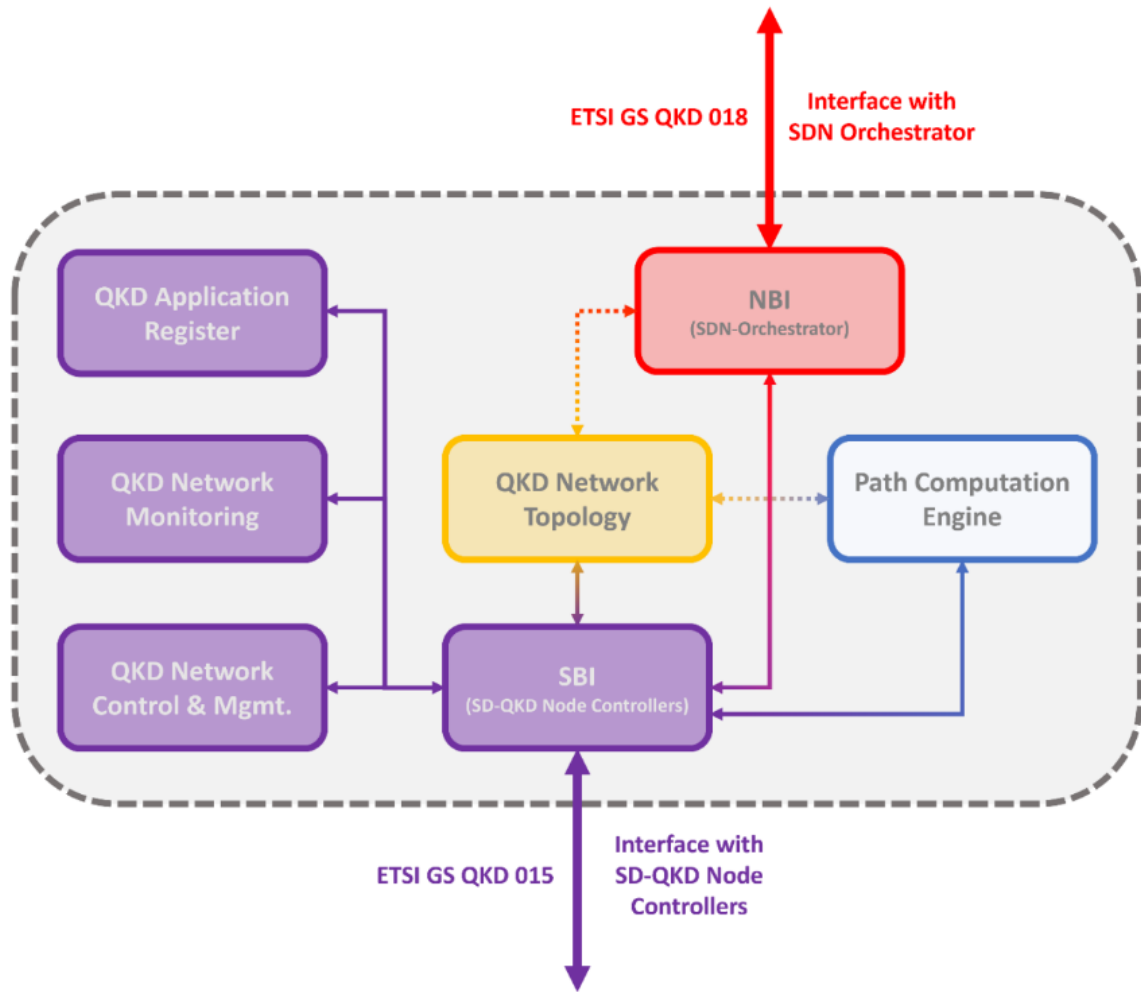


Figure 18: QKD Network Controller Architecture, Source: [52]

The architecture of the QKD Network Controller present in Figure 18 contains the following components:

- **QKD Network Topology:** Component considered as core of the controller where all data will be stored such as links, devices, services and applications. It is responsible to discover and update the QKD Network.
- **QKD Network Control & Management:** Component responsible for controlling and managing the actual network by requesting operations to the SBI which might have been requested either by the NBI or by a change requested by the administrator directly in the controller or by a change detected in the network by the controller itself. It is also responsible for provisioning tasks when a new service or an updated on an existing one is requested.
- **QKD Network Monitoring:** Component responsible for carrying out the monitoring of the QKD Net-

work. It collects all the metrics of the overall status of the QKD Network.

- QKD Application Register: Component responsible for storing and updating registered applications which are required to request QKD Keys.
- Path Computation Engine: Component responsible for computing the optimal path for the transaction of QKD Keys (might involve one or more QKD Links). This computation can have restrictions or limitation provided beforehand to be taken into account.
- SBI (QKD Node Controllers): Component responsible for the South Bound Interface which establishes a connection between the QKD Network Controller and each QKD Node (more specifically the QKD Node Controller inside the QKD Node) present in the QKD Network. From this connection, the QKD Network Controller can control and manage QKD nodes and also receive the metrics of the monitoring. This communication is supported by ETSI GS QKD 015 [40] standard.
- NBI (SDN Orchestrator): Component responsible for the North Bound Interface which establishes a connection between the QKD Network Controller and the QKD Orchestrator. From this connection, the QKD Orchestrator can orchestrate and receive monitoring metrics from the QKD Network. This communication is supported by ETSI GS QKD 018 [41] standard.

The QKD Network Controller (SDN Controller) has two different outside connections as mentioned before, the southbound one which connects the QKD Network Controller to each QKD Node and a northbound which connects the QKD Network Controller to the QKD Orchestrator. Both of these follow their respective standards that are based on YANG models.

3.4.3 QKD Orchestrator

The QKD SDN Orchestrator is responsible for the continuous automation and coordination of the available resources focusing on the most optimized way to create and deploy end-to-end services on different network domains controlled by each QKD Network Controller.

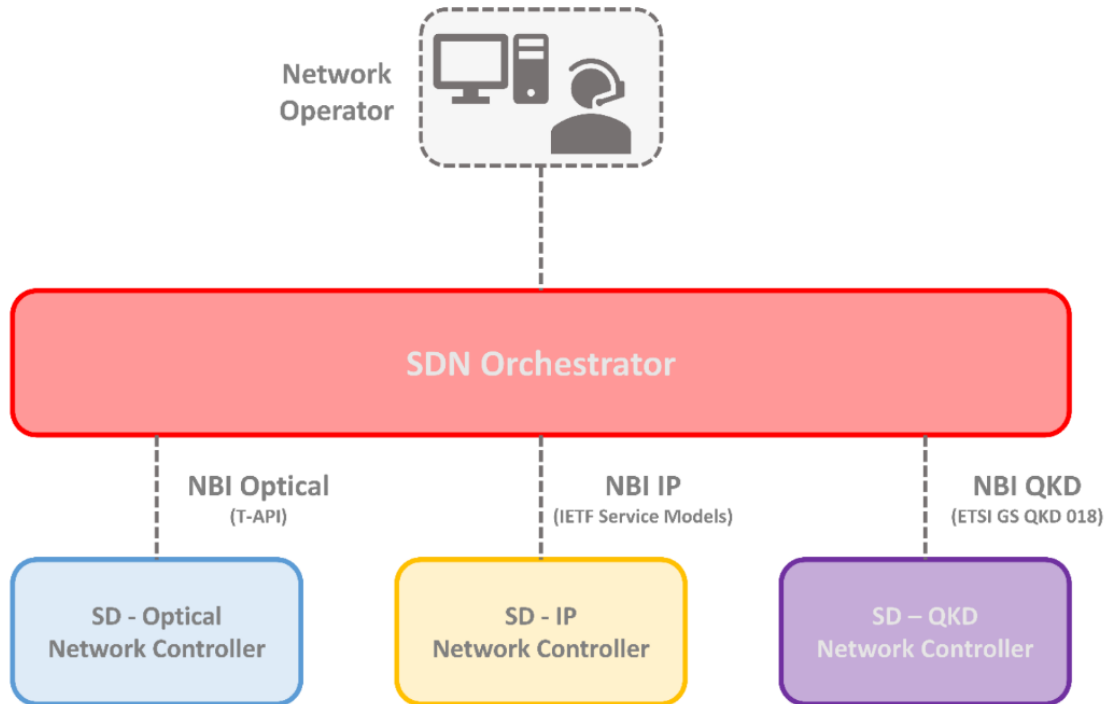


Figure 19: QKD Orchestrator, Source: [52]

The QKD SDN Orchestrator, as mentioned before, orchestrates each network domain from their respective QKD Network Controller. It will contain three interfaces one to each different network domain:

1. Optical Network
2. IP Network
3. QKD Network

However, for this scope only the interaction with the QKD Network Controller, related to the QKD Network, is important. The QKD Orchestrator will connect with the QKD Network Controller by an exposed API from the controller which should be supported by a RESTCONF NBI using the YANG models in ETSI GS QKD 018 [41] standard.

3.5 Summary

In this chapter, the design of major QKD Network components was detailed, considering the current QKD Network Architecture that was also presented. In addition, the TeraFlowSDN architecture was revisited, since it is the SDN Controller selected for the implementation of a functional prototype. Next chapter describes the implementation details.

Chapter 4

Implementation

In this chapter, all developed components are described, as well as major implementation decisions taken. The chapter starts with a description of the major developments phases, followed by the details of each component.

4.1 Introduction

The goal consists on integrating the QKD Network Controller suggested in Figure 18 into the current existing TeraflowSDN's architecture represented in Figure 5.

After studying TeraflowSDN it was possible to deduce that it follows the approach of a microservices architecture made with Python 3.x using gRPC's framework [54] which is a Remote Procedure Call (RPC) [51] to handle the communication between different components. Each component contains both its RPC Server and an RPC client, for other components to use to communicate with the server allowing each component to communicate with each other. Each component (client and server) life cycles are managed by Kubernetes [55], using the MicroK8s [56] software guaranteeing scalability and uptime.

4.2 Development Phases

This integration was planned in phases, covering the entire process from studying the requirements, planning, and development, to deployment and validation. Each phase includes a brief explanation of the choices made to ensure a smooth and effective integration.

4.2.1 Phase 1: Integration Planning

Initially, prior to the development, a planning was made for a proper integration. Started with mapping the new components from the QKD Network Controller's architecture to the current components in Ter-

aflowSDN's architecture. After reading the functionality of each components in both architectures and analyzing the present code in it, it was possible to assign most of the new components to an existing one in order to maintain a clean code and also a good structure. The next section (Components) contains a more detailed explanation about the integration of each component into TeraflowSDN.

4.2.2 Phase 2: Simulation of QKD Nodes

The purpose of this new extension to TeraflowSDN is to control QKD Nodes. For that, it is necessary to have QKD Nodes. However, since there were no real nodes for the preliminary implementation, a simulation (mock) was created. The mocked QKD Nodes simulate the behavior and implement the communication interfaces to allow for realistic interactions in controlled testing environment. This is a crucial component in the development and testing of the system. Three QKD Nodes were simulated in order to support three cases:

1. At least 1 node was necessary to prove it was possible to discover and control the QKD Node itself, executing tasks such as gathering and editing information.
2. At least 2 nodes were necessary to prove a Direct/Physic connection. This type of connection is made between two adjacent nodes.
3. At least 3 nodes were necessary to prove a Virtual connection. This type of connection is made between two non-adjacent nodes, which meant that first and last node were adjacent with the middle node but not adjacent with each other.

This simulation was made in Python 3.x with the help of a framework named Flask to create the endpoints for the communication between the TeraflowSDN and each node. Each of the simulated nodes follow the YANG model mentioned by ETSI in ETSI GS QKD 015 [40] and the respective YANG model is present in [57].

4.2.3 Phase 3: Analysis and integration of components into TeraflowSDN

Once real/simulated QKD Nodes are available, it is possible to go further with the integration.

In order to get the best outcome without having to cooperate with the existing code from TeraflowSDN, it was necessary to spend some time looking at the existing internal data structures to re-utilize the most of it without having to create more unneeded data structures. These strategies will be explained better in the next section.

After having decided on how to implement new or using the existing internal data structures. The next step is to establish the communication with the QKD Nodes.

After the communication with the nodes was possible, it was time to move on to the creation of services between them. These services require a mechanism to define a path for the service itself.

Once a service is created, it's possible to go forward with the implementation on creating an (external) application to allow the exchange of QKD Keys.

Lastly, not required for a preliminary implementation but still needed for the project itself, the monitoring of each node and the open communication between the TeraflowSDN (QKD Network Controller) and a future possible QKD Orchestrator.

4.2.4 Phase 4: Deployment in a realistic environment

This phase corresponds to the deployment of this project in a realistic environment, with real QKD Nodes. This would require new tests for the realistic environment to check if real communication was achievable and QKD Keys exchanged correctly. Since no real QKD Nodes were available, this phase was left out of the scope for this project, more specifically, the preliminary implementation.

4.2.5 Phase 5: Test and Evaluation

This phase related to the tests was only done with the simulated environment because Phase 4 wasn't achievable. Some tests were executed and they are presented in the next chapter 5.

4.3 Software Components

This section provides an overview of the essential components of the QKD Network Controller used to integrate into the existing TeraflowSDN. This process includes creating and making changes to the necessary components of TeraflowSDN in order to enable quantum key distribution in the current networks. All development was described in detail in this section.

4.3.1 QKD Network Topology

This component is responsible for storing and managing all data related to the QKD infrastructure's topology which can be integrated into the existing **Context** since it acts as the core data structure where all the information about the topology is stored. Additionally, it also allows for the management of the data

based on requests from other components. The database, already in use, is CockroachDB [58] and the connection is made with an auxiliary framework called SQLAlchemy [59].

For the integration, instead of changing the TeraflowSDN's internal database, it was decided to use a specific field in the data object named "Device" to store the new data in a JSON format. The field contains information as illustrated in Appendix B. This way, each Device's object (considered a QKD Node in the QKD Network) will hold information inside that field representing all their new data for the QKD network regarding each QKD Node itself. This new data, related to the YANG model for the SBI component, present in Appendix A, which was mentioned in ETSI GS QKD 015 [40], and the model that can be found in [57], is separated into the following 5 categories:

- **Node** - Contains information about the node id, status and version.
- **Capabilities** - Contains information about what the node is capable of, such as, key relaying.
- **Links** - Contains information about the links (virtual and direct/physic) of the node.
- **Interfaces** - Contains information about the interfaces of the node. In these interfaces it's also possible to obtain the node's endpoints.
- **Applications** - Contains information about the applications (external and internal) of the node.

4.3.2 SBI

This component is responsible for all interactions with the controlled infrastructure, namely the QKD Nodes, which can be integrated into the existing **Device** since both are related to the communication with the below level (the ones who are controlled by the SDN Controller).

In order to integrate the new component, a new class, following the abstract class provided by TeraflowSDN, was created and each method were implemented according to the needs of the QKD Network logic.

During the integration of this component, it became necessary to create a Simulated Environment (Mock) with Flask's framework to simulate the work of the QKD Nodes since there wasn't one available for testing during the development phase of the first implementation.

For now, only a RESTCONF connection which uses HTTP is available to communicate with the QKD Node but in the future it's intended to have a NETCONF as well, which uses RPC over SSH.

As mentioned in the previous component, the communication is mainly divided into 5 different categories which is shown in Appendix B.

Before and after every request, it validates that the content follows strictly the same YANG model mentioned in the previous section, with help of a library named libyang [60]. Because of the use of this library, it was possible to find out that there was a mistake in one of the YANG files from ETSI, related to the referencing of types from other YANG libraries, which was later reported on.

4.3.3 QKD Network Control & Mgmt.

This component can be integrated into the existing **Service** because both are responsible for the creation and management of the service between two different endpoints (two different QKD Nodes).

Similarly to the SBI/Device component, TeraflowSDN provides an abstract class, so a new child class was created and each method were implemented according to the needs of the QKD Network logic.

This component manages the services by interacting with the QKD Nodes through the SBI to create QKD Links either physical or virtual.

In case of a direct/physical service, it's a connection established between two adjacent QKD Nodes. This type of service is the most simple one where the TeraflowSDN will inform both QKD Nodes of the new QKD Link (direct/physical).

In case of a virtual service, the intermediary nodes are also informed about the new QKD Link (virtual), in order to be prepared for the key relaying. This type of service is more complex compared to the previous one because it requires not only configuring the intermediary nodes, but also, calculating a optimal path for the service. As the diagram in Figure 20 shows, when a new service is requested between two nodes, it will first check if there is a direct connection between those two. If they are adjacent, then it is considered a direct/physical service, and there is no need to continue with the diagram. After failing the check, it calculates the optimal path between the two nodes and later inform not only the two nodes about the new virtual service but also the intermediate nodes within the path. The path for the virtual service is computed using another component already present in TeraflowSDN.

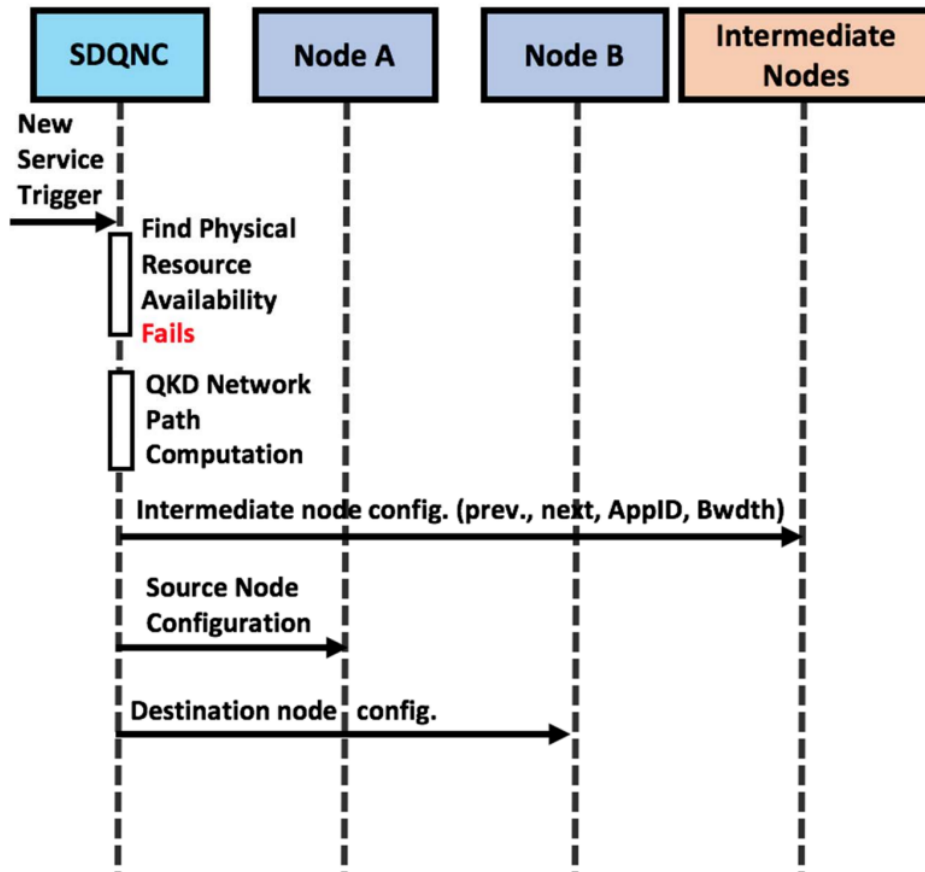


Figure 20: Sequence diagram for the dynamic creation of a virtual (multi-hop) link, Source: ETSI GS QKD 015 [40]

4.3.4 QKD Application Register

This component is new to the TeraflowSDN's architecture and, because of that, it will require the creation of a new component.

Applications are a requisite for the key exchange between two different external parties or two QKD Nodes.

There are two types of applications, one of them is external which is created when two external distinct parties request it to their respective QKD Node which will forward the request to the QKD Network Controller. For this external application to work it must be above an internal application. This second type of application, the internal one, is nothing more than a service in the TeraflowSDN vocabulary. So, when a service is created by the Service component, it also requests the Application component to create the respective internal application.

The communication for the internal application is made through RPC which is a similar implementation

in the rest of the components. However, the communication for the external application is made through a new exposed API using Flask.

Since this is a new component, it required to create a docker file for the docker image, create a CockroachDB to store information about the applications, as illustrated in Appendix C, and, also, expose the Flask's port to any external parties.

Figure 21 shows a sequence diagram of application registration in an SDN QKD Network, assuming that there is already a service established between Node A and Node B. The diagram show how external applications are requested and created for a future key exchange. First one external party (App A) requests a session (application) to its respective QKD Node (Node A) which will forward the request to the TeraflowSDN (SDQNC). Since it was the first external party to request, TeraflowSDN doesn't have information on the other external party so it will request it to wait. Once the other external party (App B) requests the session (application), it will follow the same steps as the previous one but this time TeraflowSDN will be aware of both parties and register the app to allow an exchange of keys between both parties.

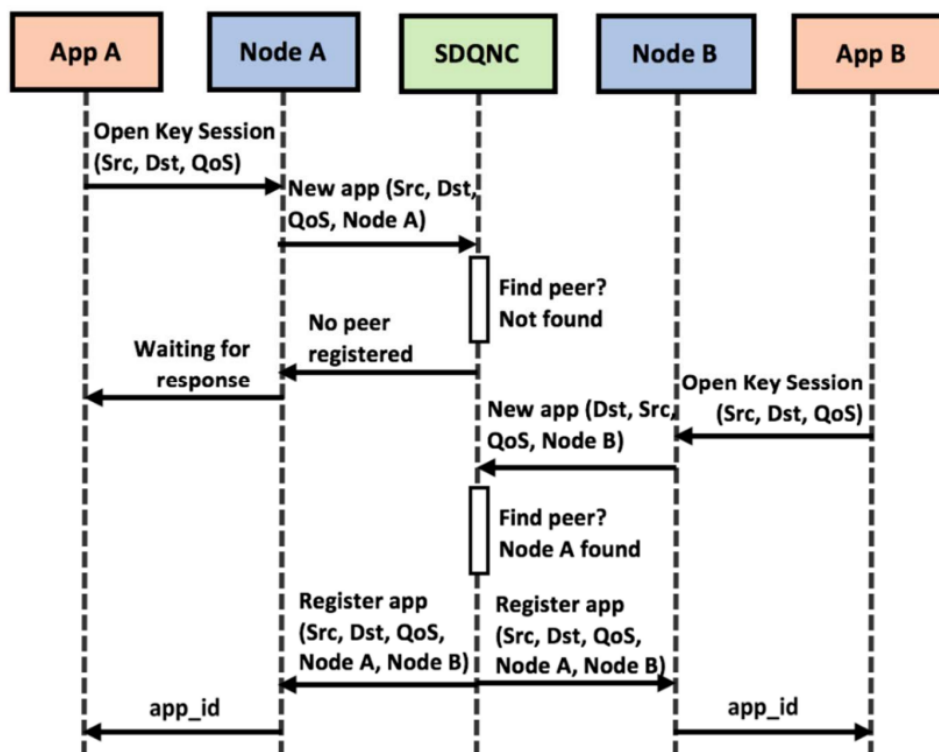


Figure 21: Sequence diagram of applications registration in an SDN QKD network, Source: ETSI GS QKD 015 [40]

4.3.5 QKD Network Monitoring

This component can be integrated into the existing **Monitoring** because both have to collect metrics from other components and the respective network itself for statistics.

This integration hasn't been on development yet because it wasn't needed in the current implementation for the QKD Network to work. However, it's expected to be only necessary to apply the same idea of monitoring of the existing IP Network but to the QKD Network with the respective required metrics.

4.3.6 Path Computation Engine

This component can be integrated into the existing **PathComp** since both are able to compute the optimal path on service request.

For the current implementation, since it still didn't have limits nor restrictions on the path computation so there was no need to develop yet a new path computation algorithm because the one in use was Shortest Path algorithm which is more than enough.

4.3.7 NBI

This component can be integrated into the existing **Compute** because both expose an API to an external system, more specifically, an SDN Orchestrator.

For the current implementation, a QKD Orchestrator hasn't been created yet so there was no need to advance with this component.

However, this component will expose an API, following the RESTCONF protocol which is based on HTTP and with the YANG models present in [61] declared by ETSI GS QKD 018 [41]. Similar to the SBI, this component will also be using a YANG Validator named libyang to validate the model in every request sent and received.

4.3.8 WebUI (Extra component)

This component, which was left in last because it's not present in the QKD Network Controller's architecture, it's important for an administrator to configure and control the QKD Network without the need of a QKD Orchestrator. For this reason, this component isn't a one time creation but a continuous development because it has to be updated upon every new feature.

As part of this integration, a new item related to QKD was added in the creation process of both the Device and Service, along with fields for the administrator to complete and fill in. Apart from it, a

new category for the QKD Application Register was, also, added to the navigation bar in order for the administrator to be able to view the current applications available, their status and the possibility to edit them. All these changes were made on Flask since it is the framework in use by TeraflowSDN. In the next chapter it's possible to observe, from the images, what was mentioned here.

4.4 Summary

In this chapter, the implementation decisions were presented and the software components detailed. Several components were created, such as simulated QKD nodes, TeraflowSDN internal structures for context information were adapted, as well as the required interfaces to interact with the QKD Nodes. Finally, the web interface was updated to facilitate the creation of testing scenarios and visualize the results. Next chapter presents a set of results.

Chapter 5

Testing and Evaluation

In this chapter, a set of use cases is presented, together with the tests performed to demonstrate them. TeraflowSDN controller, modified to include the developed QKD Network Controller is first used to load and set up a topology. New QKD devices, with direct/physical links between them, are created. Finally, a new service and an external app are setup.

5.1 Introduction

This chapter will present a brief explanation on how to use TeraflowSDN with QKD Nodes by providing the use cases which are also the test cases for the integration.

The use/tests cases will be covered in the following order:

1. Home Page (TeraflowSDN Working)
2. Load Topology from SBI into TFS using a File Descriptor
3. Create New QKD Device
4. Create New Direct/Physical Service
5. Create New Virtual Service
6. Create New External App via REST API

5.2 Use/Test Cases

This project requires real QKD Nodes to work. For this demonstration, a simulation (mock) of three QKD Nodes was implemented to allow a possible development and testing even without having real QKD Nodes.

To summarize, each simulated node is, basically, a server which stores and retrieves information about the QKD Node and acts as one.

5.2.1 Home Page (TeraflowSDN Working)

When accessing 'http://IP/webui', where the IP is the address of the server hosting TeraflowSDN, the following website should be presented, as shown in Figure 22:

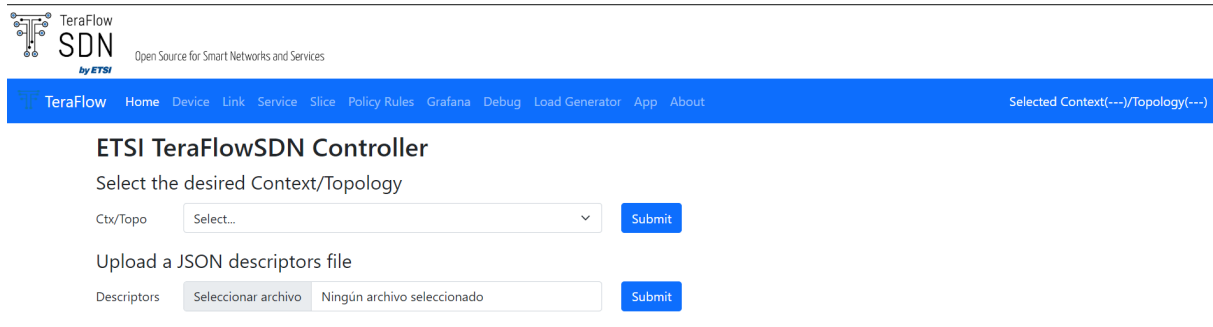


Figure 22: Homepage

5.2.2 Load Topology from SBI into TFS using a File Descriptor

The controller will be aware of the QKD Nodes by receiving the topology in a file descriptor. In Appendix D chapter there is an example of a file descriptor (JSON) containing information about three QKD Nodes (QKD1, QKD2, QKD3) with links between QKD1-QKD2 and QKD2-QKD3.

After uploading the file descriptor and later selecting the context that just appeared, the following topology will be displayed 23.

Select the desired Context/Topology

Ctx/Topo

Upload a JSON descriptors file

Descriptors

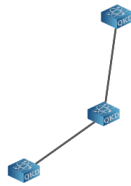


Figure 23: Homepage - Topology loaded

5.2.3 Create New QKD Device

Devices

3 devices found in context *admin*

UUID	Name	Type	Endpoints	Drivers	Status	Config Rules		
40e6c9e2-fdc8-5802-8361-413286c03494	QKD2	qkd-node	2	• QKD	ENABLED	11		
456e461e-1de7-569a-999f-73903e818e4c	QKD3	qkd-node	1	• QKD	ENABLED	9		
74520336-c12f-545e-9e18-15319f987352	QKD1	qkd-node	1	• QKD	ENABLED	9		

Figure 24: Devices

By checking the devices category, as shown in Figure 24, it's possible to observe that the creation of the QKD devices in TeraflowSDN was successful after uploading the previous file descriptor.

It's also possible to create a new device manually in TeraflowSDN by clicking in Add New Device and later selecting QKD Node which will redirect to the following page, as shown in Figure 25, for the administrator to enter more detailed information about the new QKD device.

Add New Device

ID	<input type="text"/>
Type	<input type="text" value="qkd-node"/>
Operational Status	<input type="text" value="ENABLED"/>
Drivers	<input type="checkbox"/> UNDEFINED / EMULATED <input type="checkbox"/> OPENCONFIG <input type="checkbox"/> TRANSPORT_API <input type="checkbox"/> P4 <input type="checkbox"/> IETF_NETWORK_TOPOLOGY <input type="checkbox"/> ONF_TR_532 <input type="checkbox"/> XR <input type="checkbox"/> IETF L2VPN <input type="checkbox"/> GNMI OPENCONFIG <input type="checkbox"/> FLEXSCALE <input type="checkbox"/> IETF ACTN <input type="checkbox"/> QKD
Configuration Rules	
connect/address	<input type="text" value="127.0.0.1"/>
connect/port	<input type="text" value="0"/>
connect/settings	<input type="text" value="{}"/>

Figure 25: Add new Device

Back to the devices category, in Figure 24, each device has their own configurations. Since the middle one (QKD2) has more connections (QKD1-QKD2 and QKD2-QKD3), it will, therefore, have more configuration rules compared to the others. This happens because the intermediary one hold more connections which requires more active endpoints and interfaces.

By clicking on the eye's symbols next to the device, it is possible to view their whole information and configuration as the image in Figure 26 shows. Each row on the configurations' table is a configuration rules which is any type of configuration for the Device, more specifically QKD Node, such as services' definition, connection address and port, methods of authentication, endpoints, interfaces.

Device QKD2 (40e6c9e2-fdc8-5802-8361-413286c03494)

[Back to device list](#)

[Update device](#)

[Delete device](#)

UUID: 40e6c9e2-fdc8-5802-8361-413286c03494

Name: QKD2

Type: qkd-node

Controller:

Status: ENABLED

Drivers:

- QKD

Endpoint UUID	Name	Type	Location
97b3b8e2-0e3e-5271-bc1e-ab2600b17fbd	10.211.36.220:2001	-	
bcb1cc4b-9208-54d1-bb70-8039871dd820	10.211.36.220:2002	-	

Configurations:

Key	Value		
_connect/address	• 10.211.36.220	✎	✖
_connect/port	• 22222	✎	✖
_connect/settings	• scheme: http	✎	✖
/endpoints/endpoint[10.211.36.220:2001]	• device: 10.211.36.220 • port: 2001 • uuid: 10.211.36.220:2001	✎	✖
/endpoints/endpoint[10.211.36.220:2002]	• device: 10.211.36.220 • port: 2002 • uuid: 10.211.36.220:2002	✎	✖

Figure 26: Device details

5.2.4 Create New Direct/Physical Service

A direct/physical service is established by a connection between two adjacent QKD Nodes. After heading to the Service category to create a QKD service, the user will be prompted the template illustrated in Figure 27 where it will be required to specify the name of the service and the two QKD Nodes for the service (in this case QKD1 and QKD2 which are adjacent) and their respective endpoints for the communication between the QKD Nodes. The rest of the information is optional.

TeraFlow Home Device Link Service Slice Policy Rules Grafana Debug Load Generator App About Selected Context(admin)/Topology(admin)

Add New Service [QKD]

Generic Service Parameters

Service Name: PhysService

Service Type: 6 (QKD)

Device_1: QKD1 Device_2: QKD2

Device_1 Endpoint: 10.211.36.220:1001 Device_2 Endpoint: 10.211.36.220:2001

Generic Service Constraints

Service Capacity: 10,00

Service Latency: 15,20

Service Availability:

Service Isolation: Select (Optional)

[Add New Service](#) [Cancel](#)

Figure 27: Create direct/physical Service

After creating the service, it is possible to see it and its information by going back again to Service Category. After that, as illustrated in Figure 28, it's possible to observe the newly created service with its correspondent information. The UUID is auto-generated and the status is Active which means the creation was successful.

Services

[+ Add New Service](#) 1 services found in context *admin*

UUID	Name	Type	End points	Status
b62973ad-7991-550d-ad1d-137cf79d3c40	PhysService	QKD	<ul style="list-style-type: none"> 10.211.36.220:1001 / Device: QKD1 10.211.36.220:2001 / Device: QKD2 	ACTIVE

Figure 28: Services after direct/physical service

In the Device Category, new configuration rules will be added to QKD1 and QKD2 related to the new service as the image in Figure 29 shows.





Devices

[+ Add New Device](#) 3 devices found in context *admin*

UUID	Name	Type	Endpoints	Drivers	Status	Config Rules
40e6c9e2-fdc8-5802-8361-413286c03494	QKD2	qkd-node	2	• QKD	ENABLED	13
456e461e-1de7-569a-999f-73903e818e4c	QKD3	qkd-node	1	• QKD	ENABLED	9
74520336-c12f-545e-9e18-15319f987352	QKD1	qkd-node	1	• QKD	ENABLED	11

Figure 29: Devices after direct/physical service

By clicking on the eye's symbol it's possible to see the new added configuration rules. Figure 30 shows an example of the new configuration rules added to QKD2.

/link/link[3ca90d4c-eeeb-4069-9f1e-3624f623c23b]	<ul style="list-style-type: none"> dst_interface_id: 0 dst_qkdn_id: 00000001-0000-0000-0000-000000000000 src_interface_id: 0 src_qkdn_id: 00000002-0000-0000-0000-000000000000 type: DIRECT uuid: 3ca90d4c-eeeb-4069-9f1e-3624f623c23b 		
/services/service[b62973ad-7991-550d-ad1d-137cf79d3c40]	<ul style="list-style-type: none"> qkdl_id_dst_src: 3ca90d4c-eeeb-4069-9f1e-3624f623c23b qkdl_id_src_dst: 16838eec-7760-4f63-92b7-b56eb6712bac uuid: b62973ad-7991-550d-ad1d-137cf79d3c40 		

[+ Add New Configuration](#)

Figure 30: Device details after direct/physical service

5.2.5 Create New Virtual Service

A virtual service is established by a connection between two non-adjacent QKD Nodes. Head to the Services category and create a QKD service to be prompted to the template shown in Figure 31 where it will be required to specify the name of the service and the two QKD Nodes for the service (in this case QKD1 and QKD3 which aren't adjacent) and their respective endpoints for the communication. The rest of information is optional.

The procedure is similar to the previous creation of Direct/Physical Service. The image in Figure 31 shows how to create this new Virtual Service.

Add New Service [QKD]

Generic Service Parameters

Service Name: VirtualService

Service Type: 6 (QKD)

Device_1: QKD1 Device_2: QKD3

Device_1 Endpoint: 10.211.36.220:1001 Device_2 Endpoint: 10.211.36.220:3001

Generic Service Constraints

Service Capacity: 10.00

Service Latency: 15.20

Service Availability:

Service Isolation: Select (Optional)

Figure 31: Create Virtual Service

After creating the service, it is possible to see it and its information by going back again to Service Category like in the previous creation of a direct/physical service. In the image in Figure 32, it's possible to observe again the newly created service with its correspondent information. The UUID is auto-generated and the status is Active which means the creation was successful.

Services

2 services found in context *admin*

UUID	Name	Type	End points	Status
55feca1c-1078-572b-bff6-4505fb15c0c6	VirtualService	QKD	<ul style="list-style-type: none">10.211.36.220:1001 / Device: QKD110.211.36.220:3001 / Device: QKD3	ACTIVE <input type="button" value="ⓘ"/>
b62973ad-7991-550d-ad1d-137cf79d3c40	PhysService	QKD	<ul style="list-style-type: none">10.211.36.220:1001 / Device: QKD110.211.36.220:2001 / Device: QKD2	ACTIVE <input type="button" value="ⓘ"/>

Figure 32: Services after virtual service

In the Device Category, as shown in image in Figure 33, new configuration rules will be added to QKD1 and QKD3 related to the new service. Apart from it, new rules will also be added to QKD2, even though it's not part of the specified endpoints, it's an intermediary device used for key relaying in order to allow the existence of a service between those two non-adjacent nodes.

Devices

[+ Add New Device](#) 3 devices found in context *admin*







UUID	Name	Type	Endpoints	Drivers	Status	Config Rules
40e6c9e2-fdc8-5802-8361-413286c03494	QKD2	qkd-node	2	• QKD	ENABLED	16  
456e461e-1de7-569a-999f-73903e818e4c	QKD3	qkd-node	1	• QKD	ENABLED	12  
74520336-c12f-545e-9e18-15319f987352	QKD1	qkd-node	1	• QKD	ENABLED	13  

Figure 33: Devices after virtual service

By clicking on the eye's symbols we can see the new added configuration rules. Figure 34 shows an example of the new configuration rules added to QKD2 which is the middle one between the connection. Even though it's a connection between QKD1 and QKD3, QKD2 will also contain all the information about the service.

`/link/link[e9dc2fc8-0ea6-4957-8d64-f8969aa42ff1]`

- `dst_interface_id:` 0
- `dst_qkdn_id:` 00000003-0000-0000-0000-000000000000
- `src_interface_id:` 0
- `src_qkdn_id:` 00000001-0000-0000-0000-000000000000
- `type:` VIRTUAL
- `uuid:` e9dc2fc8-0ea6-4957-8d64-f8969aa42ff1
- `virt_bandwidth:` 0
- `virt_next_hops:` [00000003-0000-0000-0000-000000000000]
- `virt_prev_hop:` 00000001-0000-0000-0000-000000000000

`/link/link[8fa7f028-8b05-41af-a606-13d73104f713]`

- `dst_interface_id:` 0
- `dst_qkdn_id:` 00000001-0000-0000-0000-000000000000
- `src_interface_id:` 0
- `src_qkdn_id:` 00000003-0000-0000-0000-000000000000
- `type:` VIRTUAL
- `uuid:` 8fa7f028-8b05-41af-a606-13d73104f713
- `virt_bandwidth:` 0
- `virt_next_hops:` [00000001-0000-0000-0000-000000000000]
- `virt_prev_hop:` 00000003-0000-0000-0000-000000000000

`/services/service[55fec1c-1078-572b-bff6-4505fb15c0c6]`

- `qkdl_id_dst_src:` 8fa7f028-8b05-41af-a606-13d73104f713
- `qkdl_id_src_dst:` e9dc2fc8-0ea6-4957-8d64-f8969aa42ff1
- `uuid:` 55fec1c-1078-572b-bff6-4505fb15c0c6

Figure 34: Device details after virtual service

When a virtual service is created, two virtual links are created (one in each direction), and, each virtual link must have its own, respective, internal application. In the app category it's possible to observe the new internal app (one for each virtual link).

Apps

2 apps found in context *admin*

UUID	Status	Type	Device 1	Device 2
81375a52-0b71-4116-9e78-aa54681639d8	ON	INTERNAL	• QKD3	• QKD1
fe89ac64-1d66-4083-a556-fc6dcf05d3f4	ON	INTERNAL	• QKD1	• QKD3

Figure 35: Apps after virtual service

5.2.6 Create New External App via REST API

Apart from the internal applications which come with the virtual links. There are also external application to make the exchange of QKD Keys between two external parties possible. For this to happen, two external entities must request a connection using an API request as shown in Figure 36.

```
>>> import requests
>>> requests.post('http://10.211.36.220/app/create_qkd_app', json={'app': {'server_app_id': '1', 'client_app_id': [], 'app_status': 'ON', 'local_qkdn_id': '00000001-0000-0000-0000-000000000000', 'backing_qkdl_id': ['00000003-0002-0000-0000-000000000000']}})
<Response [200]>
>>> requests.post('http://10.211.36.220/app/create_qkd_app', json={'app': {'server_app_id': '1', 'client_app_id': [], 'app_status': 'ON', 'local_qkdn_id': '00000003-0000-0000-0000-000000000000', 'backing_qkdl_id': ['00000003-0002-0000-0000-000000000000']}})
<Response [200]>
```

Figure 36: Request external App

If only one party request, the app will appear but with only the QKD Node 1. After both parties request it, it will look like the image shown in Figure 37, which corresponds to the communication between the QKD Nodes and the QKD SDN Controller represented by the diagram in Figure 21.

Apps

3 apps found in context *admin*

UUID	Status	Type	Device 1	Device 2
81375a52-0b71-4116-9e78-aa54681639d8	ON	INTERNAL	• QKD3	• QKD1
8cbf12e2-19f9-4f5c-a805-ef172c14b364	ON	EXTERNAL	• QKD1	• QKD3
fe89ac64-1d66-4083-a556-fc6dcf05d3f4	ON	INTERNAL	• QKD1	• QKD3

Figure 37: Apps after external app

5.3 Summary

This was successfully tested with simulated QKD Nodes to demonstrate the functionality of the newly integrated software components in TeraflowSDN. As helpful as these simulations are, future testing needs to get carried out using real-world scenarios which will further validate the performance and capabilities of such a system.

Chapter 6

Conclusions and future work

In this chapter, some conclusions were summarized from this research and prospects of future work were presented, highlighting the importance of QKD and SDN for future networks.

6.1 Conclusions

This dissertation explored the integration of the QKD Network Controller into the existing TeraflowSDN (SDN Controller), proving, through a simulated environment, that it is capable of controlling QKD Nodes using the protocols defined by ETSI.

All the steps mentioned in the Objectives section were fulfilled in order to execute the proposed project.

It began with the study of the State of the Art, which was conducted to provide the necessary context and understanding of the theme of this dissertation, specifically, knowledge about 6G, QKD, and SDN.

This was followed by an analysis of how the TeraflowSDN (the chosen SDN Controller) is architected and how it works internally. An analysis was also made on how the entire QKD Network is architected, focusing on the various aspects of each internal component (QKD Node, QKD Network Controller, QKD Orchestrator).

The objective of this development was to integrate the architecture of the QKD Network Controller into the existing TeraflowSDN (SDN Controller), which was explained by presenting the planning process along with each modification and addition, as well as the technologies that were used.

Finally, it was demonstrated how to use the existing TeraflowSDN with this new integration. It was possible to observe the TeraflowSDN being initiated, followed by the loading of the QKD Nodes topology into it. Subsequently, it was demonstrated how to manually declare a new QKD Node in TeraflowSDN, followed by the creation of a service between QKD Nodes. Both types of services (direct/physical and virtual) were created, and an external application to provide QKD Keys between two parties was also requested.

In conclusion, all the proposed steps in the present project were successfully executed, leading to the conclusion that it is possible to integrate the suggested QKD Network Controller into an existing SDN Controller (in this case, TeraflowSDN), with this model functioning effectively.

This dissertation demonstrated that the presented project, regarding the proposed integration, appears as a possible implementation of this new concept of the quantum world into current systems.

6.2 Prospect for future work

The research conducted in this dissertation provides a solid idea and preliminary development for the integration. However, there are still some tasks to be carried out, such as:

- Testing with real QKD Nodes. It should be noted that so far it has only been possible to test with simulated QKD Nodes, so it is necessary to proceed with testing in real ones. To integrate them adequately into the real world, it is necessary to test them first in a real scenario, which involves the use of real QKD Nodes, which are still under development.
- Need for monitoring. This development only communicates with the simulated QKD Nodes to control them. However, in the future, the controller should also be able to monitor them in order to obtain statistics and better performance.
- Need for new path algorithm optimization. Currently, this development considers paths between QKD Nodes without limitations or restrictions, using a simple path computation algorithm called the shortest path. However, in the future, a new algorithm should be developed to allow for these limitations and restrictions in order to calculate the ideal path for a service, optimizing the result.
- Need for the creation of a QKD Orchestrator. In the future, the **Opensec** project aims to develop the QKD Orchestrator to orchestrate the QKD Network. For this to be possible, the QKD Network Controller must expose an NBI to communicate with the upper level, the QKD Orchestrator.

Bibliography

- [1] CTTC. 6g-opensec_keys - quantum key distribution for security in open and disaggregated 6g networks, 2021. <https://www.cttc.cat/project/quantum-key-distribution-for-security-in-open-and-disaggregated-6g-networks/> [Accessed: 01/09/2024].
- [2] CTTC. 6g-opensec_security - secure network slice manager for open and disaggregated 6g networks, 2021. <https://www.cttc.cat/project/secure-network-slice-manager-for-open-and-disaggregated-6g-networks/> [Accessed: 01/09/2024].
- [3] CTTC. 6g-opensec_trust - dlt-based trust management for open and disaggregated 6g networks, 2021. <https://www.cttc.cat/project/dlt-based-trust-management-for-open-and-disaggregated-6g-networks/> [Accessed: 01/09/2024].
- [4] DNAC. Nof 2024: 15th international conference on network of the future, 2024. <https://nof.dnac.org/> [Accessed: 01/09/2024].
- [5] ETSI. Etsi teraflowsdn controller repository, 2022. <https://labs.etsi.org/rep/tfs/controller/-/tree/master> [Accessed: 01/09/2024].
- [6] Optare Solutions. Optare solutions, 2002. <https://optaresolutions.com/en/> [Accessed: 29/05/2024].
- [7] Walid Saad, Mehdi Bennis, and Mingzhe Chen. A vision of 6g wireless systems: Applications, trends, technologies, and open research problems. *IEEE Network*, 34(3):134–142, 2020. doi: 10.1109/MNET.001.1900287.
- [8] Shuping Dang, Osama Amin, Basem Shihada, and Mohamed-Slim Alouini. What should 6g be? *Nature Electronics*, 3(1):20–29, January 2020. ISSN 2520-1131. doi: 10.1038/s41928-019-0355-6. URL <http://dx.doi.org/10.1038/s41928-019-0355-6>.
- [9] Samar Elmeadawy and Raed M. Shubair. 6g wireless communications: Future technologies and

- research challenges. In *2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, pages 1–5, 2019. doi: 10.1109/ICECTA48151.2019.8959607.
- [10] Yuan Cao, Yongli Zhao, Qin Wang, Jie Zhang, Soon Xin Ng, and Lajos Hanzo. The evolution of quantum key distribution networks: On the road to the qinternet. *IEEE Communications Surveys & Tutorials*, 24(2):839–894, 2022. doi: 10.1109/COMST.2022.3144219.
- [11] Nicolas Sendrier. Code-based cryptography: State of the art and perspectives. *IEEE Security & Privacy*, 15(4):44–50, 2017. doi: 10.1109/MSP.2017.3151345.
- [12] Denis Butin. Hash-based signatures: State of play. *IEEE Security & Privacy*, 15(4):37–43, 2017. doi: 10.1109/MSP.2017.3151334.
- [13] Hamid Nejatollahi, Nikil Dutt, Sandip Ray, Francesco Regazzoni, Indranil Banerjee, and Rosario Cammarota. Post-quantum lattice-based cryptography implementations: A survey. *ACM Comput. Surv.*, 51(6), jan 2019. ISSN 0360-0300. doi: 10.1145/3292548. URL <https://doi.org/10.1145/3292548>.
- [14] Jintai Ding and Albrecht Petzoldt. Current state of multivariate cryptography. *IEEE Security & Privacy*, 15(4):28–36, 2017. doi: 10.1109/MSP.2017.3151328.
- [15] Paul Adrien Maurice Dirac. *The Principles of Quantum Mechanics*. Clarendon Press, Oxford,, 1930.
- [16] W. K. Wootters and W. H. Zurek. A single quantum cannot be cloned. *Nature*, 299(5886):802–803, Oct 1982. ISSN 1476-4687. doi: 10.1038/299802a0. URL <https://doi.org/10.1038/299802a0>.
- [17] Chonggang Wang and Akbar Rahman. Quantum-enabled 6g wireless networks: Opportunities and challenges. *IEEE Wireless Communications*, 29(1):58–69, 2022. doi: 10.1109/MWC.006.00340.
- [18] Diego Kreutz, Fernando M. V. Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015. doi: 10.1109/JPROC.2014.2371999.
- [19] Rashid Amin, Martin Reisslein, and Nadir Shah. Hybrid sdn networks: A survey of existing approaches. *IEEE Communications Surveys & Tutorials*, 20(4):3259–3306, 2018. doi: 10.1109/COMST.2018.2837161.

- [20] Silviu-Gabriel Topoloi and Eugen Borcoci. Software defined networking and network function virtualisation cooperation-experiments. In *2018 International Conference on Communications (COMM)*, pages 281–286, 2018. doi: 10.1109/ICComm.2018.8484739.
- [21] Hamid Nejatollahi, Nikil Dutt, Sandip Ray, Francesco Regazzoni, Indranil Banerjee, and Rosario Cammarota. Post-quantum lattice-based cryptography implementations: A survey. *ACM Comput. Surv.*, 51(6), jan 2019. ISSN 0360-0300. doi: 10.1145/3292548. URL <https://doi.org/10.1145/3292548>.
- [22] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Raouf Boutaba. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 18(1):236–262, 2016. doi: 10.1109/COMST.2015.2477041.
- [23] A. Marsico, A. Reid, F.J. Ramón, and G. García. Osm in action, 2021. <https://forge.etsi.org/rep/qkd/gS015-ctrl-int-sd> [Accessed: 29/05/2024].
- [24] Shuo Wang, Xing Zhang, Yan Zhang, Lin Wang, Juwo Yang, and Wenbo Wang. A survey on mobile edge networks: Convergence of computing, caching and communications. *IEEE Access*, 5:6757–6779, 2017. doi: 10.1109/ACCESS.2017.2685434.
- [25] Jhon Javier Loza Llucó. Evaluación y comparativa de controladores sdn para despliegues de redes 5g. *RiuNet*, 2023.
- [26] Jan Medved, Robert Varga, Anton Tkacik, and Ken Gray. Opendaylight: Towards a model-driven sdn controller architecture. In *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, pages 1–6, 2014. doi: 10.1109/WoWMoM.2014.6918985.
- [27] Ameer Sameer. Open network operating system (onos). In *Open Network Operating System (ONOS)*, 12 2015. doi: 10.13140/RG.2.1.3394.5689.
- [28] *Slice Grouping for Transport Network Slices Using Hierarchical Multi-domain SDN Controllers*, June 2023. Zenodo. doi: 10.1364/OFC.2023.M3Z.9. URL <https://doi.org/10.1364/OFC.2023.M3Z.9>.
- [29] Vamsi Krishna Patchava Mansi Dikshit Revathi Venkataraman Vaishnavi Moorthy. Technical comparative analysis between odl and onos software defined controllers. *International Journal of Control Theory and Applications*, 10, 2017.

- [30] Omer Bulakci. *Towards sustainable and trustworthy 6G: Challenges, enablers, and architectural design*. Now Publishers, 2023.
- [31] Ricard Vilalta, Raul Muñoz, Ramon Casellas, Ricardo Martínez, Victor López, Oscar González de Dios, Antonio Pastor, Georgios P. Katsikas, Felix Klaedtke, Paolo Monti, Alberto Mozo, Thomas Zinner, Harald Øverby, Sergio Gonzalez-Diaz, Håkon Lønsethagen, José-Miguel Pulido, and Daniel King. Teraflow: Secured autonomic traffic management for a tera of sdn flows. In *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, pages 377–382, 2021. doi: 10.1109/EuCNC/6GSummit51104.2021.9482469.
- [32] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. Devops. *IEEE Software*, 33(3):94–100, 2016. doi: 10.1109/MS.2016.68.
- [33] ETSI. Teraflowsdn: B5g network scenario, 2023. <https://www.teraflow-h2020.eu/scenarios> [Accessed: 29/05/2024].
- [34] Hua Wang, Yongli Zhao, and Avishek Nag. Quantum-key-distribution (qkd) networks enabled by software-defined networks (sdn). *Applied Sciences*, 9(10), 2019. ISSN 2076-3417. doi: 10.3390/app9102081. URL <https://www.mdpi.com/2076-3417/9/10/2081>.
- [35] Charles H. Bennett and Gilles Brassard. Quantum cryptography: Public key distribution and coin tossing. *Theoretical Computer Science*, 560:7–11, December 2014. ISSN 0304-3975. doi: 10.1016/j.tcs.2014.05.025. URL <http://dx.doi.org/10.1016/j.tcs.2014.05.025>.
- [36] Yuan Cao, Yongli Zhao, Xiaosong Yu, Lijie Cheng, Ziqin Li, Guojun Liu, and Jie Zhang. Experimental demonstration of end-to-end key on demand service provisioning over quantum key distribution networks with software defined networking. In *2019 Optical Fiber Communications Conference and Exhibition (OFC)*, pages 1–3, 2019.
- [37] Yuan Cao, Yongli Zhao, Carlos Colman-Meixner, Xiaosong Yu, and Jie Zhang. Key on demand (kod) for software-defined optical networks secured by quantum key distribution (qkd). *Opt. Express*, 25(22):26453–26467, Oct 2017. doi: 10.1364/OE.25.026453. URL <https://opg.optica.org/oe/abstract.cfm?URI=oe-25-22-26453>.
- [38] R. S. Tessinari, R. I. Woodward, and A. J. Shields. Software-defined quantum network using a qkd-secured sdn controller and encrypted messages, 2023.

- [39] ETSI. ETSI GS QKD 014 v1.1.1. Standard, European Telecommunications Standards Institute, 650 Route des Lucioles F-06921 Sophia Antipolis Cedex - FRANCE, February 2019. URL https://www.etsi.org/deliver/etsi_gs/QKD/001_099/014/01.01.01_60/gs_qkd014v010101p.pdf.
- [40] ETSI. ETSI GS QKD 015 v2.1.1. Standard, European Telecommunications Standards Institute, 650 Route des Lucioles F-06921 Sophia Antipolis Cedex - FRANCE, April 2022. URL https://www.etsi.org/deliver/etsi_gs/QKD/001_099/015/02.01.01_60/gs_QKD015v020101p.pdf.
- [41] ETSI. ETSI GS QKD 018 v1.1.1. Standard, European Telecommunications Standards Institute, 650 Route des Lucioles F-06921 Sophia Antipolis Cedex - FRANCE, April 2022. URL https://www.etsi.org/deliver/etsi_gs/QKD/001_099/018/01.01.01_60/gs_qkd018v010101p.pdf.
- [42] Alejandro Aguado, V. Martin, D. Lopez, M. Peev, J. Martinez-Mateo, J.L. Rosales, F. de la Iglesia, M. Gomez, Emilio Hugues Salas, A. Lord, Reza Nejabati, and Dimitra Simeonidou. Quantum-aware software defined networks. In *6th International Conference on Quantum Cryptography (QCRYPT 2016)*. QCrypt, September 2016. 42nd European Conference on Optical Communication, ECOC 2016 : 42nd European Conference and Exhibition on Optical Communication ; Conference date: 18-09-2016 Through 22-09-2016.
- [43] Wanrong Yu, Baokang Zhao, and Zhe Yan. Software defined quantum key distribution network. In *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, pages 1293–1297, 2017. doi: 10.1109/CompComm.2017.8322751.
- [44] Zhao, Yongli and Cao, Yuan and Wang, Wei and Wang, Hua and Yu, Xiaosong and Zhang, Jie and Tornatore, Massimo and Wu, Yu and Mukherjee, and Biswanath. Resource allocation in optical networks secured by quantum key distribution. *IEEE Communications Magazine*, 56(8):130–137, 2018. doi: 10.1109/MCOM.2018.1700656.
- [45] TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU. Quantum key distribution networks – Control and management . Standard, International Telecommunication Union, September 2020. URL <https://www.itu.int/rec/T-REC-Y.3804-202009-I/en>.
- [46] TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU. Quantum key distribution networks

- Key management . Standard, International Telecommunication Union, December 2020. URL <https://www.itu.int/rec/T-REC-Y.3803-202012-I/en>.
- [47] Martin Björklund. YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020, October 2010. URL <https://www.rfc-editor.org/info/rfc6020>.
- [48] Andy Bierman, Martin Björklund, and Kent Watsen. RESTCONF Protocol. RFC 8040, January 2017. URL <https://www.rfc-editor.org/info/rfc8040>.
- [49] Rob Enns, Martin Björklund, Andy Bierman, and Jürgen Schönwälder. Network Configuration Protocol (NETCONF). RFC 6241, June 2011. URL <https://www.rfc-editor.org/info/rfc6241>.
- [50] ETSI. Teraflowsdn: Architecture, 2023. <https://www.teraflow-h2020.eu/technologies> [Accessed: 29/05/2024].
- [51] R. Thurlow. RPC: Remote Procedure Call Protocol Specification Version 2. RFC 5531 (Draft Standard), May 2009. URL <http://www.ietf.org/rfc/rfc5531.txt>.
- [52] Juan Morales Sáez and Antonio Pastor Perales. Architecture and definition of interfaces for quantum keys distribution in 6g network. In *Derivable E6*. Project 6G-OPENSEC-KEYS, September 2023.
- [53] ETSI. ETSI GS QKD 004 v2.1.1. Standard, European Telecommunications Standards Institute, 650 Route des Lucioles F-06921 Sophia Antipolis Cedex - FRANCE, August 2020. URL https://www.etsi.org/deliver/etsi_gs/QKD/001_099/004/02.01.01_60/gs_QKD004v020101p.pdf.
- [54] grpc. grpc –an rpc library and framework, 2016. <https://github.com/grpc/grpc> [Accessed: 29/05/2024].
- [55] Brendan Burns, Joe Beda, Kelsey Hightower, and Lachlan Evenson. *Kubernetes: up and running.* ” O’Reilly Media, Inc.”, 2022.
- [56] Canonical. Microk8s, 2022. <https://microk8s.io/> [Accessed: 29/05/2024].
- [57] ETSI. Qkd control interface for sdn, 2022. <https://forge.etsi.org/rep/qkd/gs015-ctrl-int-sd> [Accessed: 29/05/2024].
- [58] Rebecca Taft, Irfan Sharif, Andrei Matei, Nathan VanBenschoten, Jordan Lewis, Tobias Grieger, Kai Niemi, Andy Woods, Anne Birzin, Raphael Poss, Paul Bardea, Amruta Ranade, Ben Darnell, Bram

Gruneir, Justin Jaffray, Lucy Zhang, and Peter Mattis. Cockroachdb: The resilient geo-distributed sql database. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 1493–1509, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367356. doi: 10.1145/3318464.3386134. URL <https://doi.org/10.1145/3318464.3386134>.

[59] James Gardner. *Introducing the Model and SQLAlchemy*, chapter -1, pages –1. Apress, 01 2009. ISBN 978-1-59059-934-1. doi: 10.1007/978-1-4302-0534-0_7.

[60] CESNET. Yang data modeling language library, 2016. <https://github.com/CESNET/libyang> [Accessed: 29/05/2024].

[61] ETSI. Qkd orchestration interface for sdn, 2022. <https://forge.etsi.org/rep/qkd/g018-orch-int-sdn> [Accessed: 29/05/2024].

Part I

Appendices

Appendix A

The SBI as defined in ETSI QKD 015 v2.1.1

The SBI from ETSI GS QKD 015 v2.1.1 uses the following YANG Protocol:

```
module: etsi-qkd-sdn-node
  +--rw qkd_node
  +--rw qkdn_id yang:uuid
  +--ro qkdn_status? etsi-qkdn-types:qkdn-status-types
  +--rw qkdn_version? string
  +--rw qkdn_location_id? string
  +--rw qkdn_capabilities
    +--rw link_stats_support? boolean
    +--rw application_stats_support? boolean
    +--rw key_relay_mode_enable? boolean
  +--rw qkdn_apps
    +--rw qkd_app* [app_id]
      +--rw app_id yang:uuid
      +--ro app_status? etsi-qkdn-types:app-status-types
      +--rw app_type? etsi-qkdn-types:qkd-app-types
      +--rw server_app_id? inet:uri
      +--rw client_app_id* inet:uri
      +--rw app_priority? uint32
      +--ro creation_time? yang:date-and-time
      +--rw expiration_time? yang:date-and-time
      +--rw app_statistics
        +--ro statistics* [end_time]
          +--ro end_time yang:date-and-time
          +--ro start_time? yang:date-and-time
          +--ro consumed_bits? uint32
      +--rw app_qos
        +--rw max_bandwidth? uint32
        +--rw min_bandwidth? uint32
        +--rw jitter? uint32
        +--rw ttl? uint32
        +--rw clients_shared_path_enable? boolean
        +--rw clients_shared_keys_required? boolean
    +--rw backing_qkdl_id* yang:uuid
    +--rw local_qkdn_id? yang:uuid
    +--rw remote_qkdn_id? yang:uuid
```

```

+--rw qkd_interfaces
  +--rw qkd_interface* [qkdi_id]
    +--rw qkdi_id uint32
    +--rw qkdi_model? string
    +--rw qkdi_type? etsi-qkdn-types:qkd-technology-types
    +--rw qkdi_att_point
      +--rw device? string
      +--rw port? uint32
    +--rw qkdi_capabilities
      +--rw role_support? etsi-qkdn-types:qkd-role-types
      +--rw wavelength_range? etsi-qkdn-types:wavelength-range-type
      +--rw max_absorption? decimal64
    +--ro qkdi_status? etsi-qkdn-types:iface-status-types
+--rw qkd_links
  +--rw qkd_link* [qkdl_id]
    +--rw qkdl_id yang:uuid
    +--rw qkdl_status? etsi-qkdn-types:link-status-types
    +--rw qkdl_enable? boolean
    +--rw qkdl_local
      +--rw qkdn_id? yang:uuid
      +--rw qkdi_id? uint32
    +--rw qkdl_remote
      +--rw qkdn_id? yang:uuid
      +--rw qkdi_id? uint32
    +--rw qkdl_type? etsi-qkdn-types:qkd-link-types
    +--rw qkdl_applications* yang:uuid
    +--rw virt_prev_hop? yang:uuid
    +--rw virt_next_hop* yang:uuid
    +--rw virt_bandwidth? uint32
    +--rw phys_channel_att? decimal64
    +--rw phys_wavelength? etsi-qkdn-types:wavelength
    +--rw phys_qkd_role? etsi-qkdn-types:qkd-role-types
    +--rw qkdl_performance
      +--ro expected_consumption? uint32
      +--ro skr? uint32
      +--ro eskr? uint32
      +--ro phys_perf* [perf_type]
        +--ro perf_type etsi-qkdn-types:phys-perf-types
        +--ro value? decimal64

```

Appendix B

QKD Network topology stored in TeraflowSDN

The current database that holds most of the information relevant to TeraflowSDN is present in the context component. It holds information about Devices, Links, Services, etc. For the current development, all information about a QKD Node such as detailed information on links, services, apps, interfaces, endpoints, services and the node itself are stored in the respective device's configuration rules like follows:

```
{
  resource_key:  : /node,
  resource_value : {
    "qkdn_id":UUID,
    "qkdn_status":STATUS,
    ...
  }
}

{
  resource_key:  : /capabilities,
  resource_value : {
    "link_stats_support":BOOL,
    "application_stats_support":BOOL,
    "key_relay_mode_enable":BOOL
  }
}

{
  resource_key:  : /app[UUID],
  resource_value : {
    "app_id":UUID,
    "app_status":STATUS,
    "app_type":TYPE,
    ...
  }
}

{
  resource_key:  : /endpoints/endpoint [UUID],
  resource_value : {
```

```

    "uuid":UUID,
    "device":IP,
    "port":PORT
  }
}

{
  resource_key:  : /link[UUID],
  resource_value : {
    "uuid":UUID,
    "type":TYPE,
    ...
  }
}

{
  resource_key:  : /interface[ID or IP:PORT],
  resource_value : {
    "qkdi_id":ID,
    "qkdi_type":TYPE, "qkdi_att_point": {
      "device":IP,
      "port":PORT
    },
    ...
  }
}

{
  resource_key:  : /service[UUID],
  resource_value : {
    "uuid":UUID,
    "qkdl_id_dst_src":UUID,
    "qkdl_id_src_dst":UUID
  }
}

```

Appendix C

Database schema for new App Component

Apart from what was already present in TeraflowSDN, a new database (CockroachDB) was created to support the new component App with the following schema:

Name	Type
app_uuid (PK)	UUID
context_uuid	UUID
app_status	QKAppStatusEnum
app_type	QKAppTypesEnum
server_app_id	String
client_app_id	Array(String)
backing_qkdl_uuid	Array(UUID)
local_device_uuid	UUID
remote_device_uuid	UUID
created_at	DateTime
updated_at	DateTime

Appendix D

Example Network Topology in JSON format

The following data is respective to the information passed to the TeraflowSDN in a JSON descriptor file on a loading stage.

```
{
  "contexts": [
    {"context_id": {"context_uuid": {"uuid": "UUID"}}}
  ],
  "topologies": [
    {"topology_id": {
      "topology_uuid": {"uuid": "UUID"},
      "context_id": {"context_uuid": {"uuid": "UUID"}}
    }}
  ],
  "devices": [
    {
      "device_id": {"device_uuid": {"uuid": "QKD1"}},
      "device_type": "qkd-node",
      "device_operational_status": 0, "device_drivers": [11], "device_endpoints": [],
      "device_config": {"config_rules": [
        {"action": 1, "custom": {
          "resource_key": "_connect/address", "resource_value": "IP1"
        }},
        {"action": 1, "custom": {
          "resource_key": "_connect/port", "resource_value": "PORT1"
        }},
        {"action": 1, "custom": {
          "resource_key": "_connect/settings", "resource_value": {"scheme": "http"}
        }}
      ]}
    },
    {
      "device_id": {"device_uuid": {"uuid": "QKD2"}},
      "device_type": "qkd-node",
      "device_operational_status": 0, "device_drivers": [11], "device_endpoints": [],
      "device_config": {"config_rules": [
        {"action": 1, "custom": {
          "resource_key": "_connect/address", "resource_value": "IP2"
        }},
        {"action": 1, "custom": {
          "resource_key": "_connect/port", "resource_value": "PORT2"
        }},
        {"action": 1, "custom": {
          "resource_key": "_connect/settings", "resource_value": {"scheme": "http"}
        }}
      ]}
    }
  ]
}
```

```

},
{
  "device_id": {"device_uuid": {"uuid": "QKD3"}},
  "device_type": "qkd-node",
  "device_operational_status": 0, "device_drivers": [11], "device_endpoints": [],
  "device_config": {"config_rules": [
    {"action": 1, "custom": {
      "resource_key": "_connect/address", "resource_value": "IP3"
    }},
    {"action": 1, "custom": {
      "resource_key": "_connect/port", "resource_value": "PORT3"
    }},
    {"action": 1, "custom": {
      "resource_key": "_connect/settings", "resource_value": {"scheme": "http"}
    }}
  ]}
}
],
"links": [
  {
    "link_id": {"link_uuid": {"uuid": "QKD1/IP1:PORT1==QKD2/IP2:PORT2"}},
    "link_endpoint_ids": [
      {"device_id": {
        "device_uuid": {"uuid": "QKD1"}}, "endpoint_uuid": {"uuid": "IP1:PORT1"}
      },
      {"device_id": {
        "device_uuid": {"uuid": "QKD2"}}, "endpoint_uuid": {"uuid": "IP2:PORT2"}
      }
    ]
  }
],
{
  "link_id": {"link_uuid": {"uuid": "QKD2/IP2:PORT2==QKD1/IP1:PORT1"}},
  "link_endpoint_ids": [
    {"device_id": {
      "device_uuid": {"uuid": "QKD2"}}, "endpoint_uuid": {"uuid": "IP2:PORT2"}
    },
    {"device_id": {
      "device_uuid": {"uuid": "QKD1"}}, "endpoint_uuid": {"uuid": "IP1:PORT1"}
    }
  ]
},
{
  "link_id": {"link_uuid": {"uuid": "QKD2/IP2:PORT2==QKD3/IP3:PORT3"}},
  "link_endpoint_ids": [
    {"device_id": {
      "device_uuid": {"uuid": "QKD2"}}, "endpoint_uuid": {"uuid": "IP2:PORT2"}
    },
    {"device_id": {
      "device_uuid": {"uuid": "QKD3"}}, "endpoint_uuid": {"uuid": "IP3:PORT3"}
    }
  ]
},
{
  "link_id": {"link_uuid": {"uuid": "QKD3/IP3:PORT3==QKD2/IP2:PORT2"}},
  "link_endpoint_ids": [
    {"device_id": {
      "device_uuid": {"uuid": "QKD3"}}, "endpoint_uuid": {"uuid": "IP3:PORT3"}
    }
  ],
}

```

```
    {"device_id": {
      "device_uuid": {"uuid": "QKD2"}}, "endpoint_uuid": {"uuid": "IP2:PORT2"}
    }
  ]
}
}
```


This work has been funded by the "Ministerio de Asuntos Económicos y Transformación Digital" and the European Union-NextGenerationEU within the framework of the "Plan de Recuperación, Transformación y Resiliencia" and the "Mecanismo de Recuperación y Resiliencia" under reference TSI-063000-2021-61 (6G-OPENSEC KEYS).



Universidade do Minho

