Service for Deploying Digital Twins of QKD Networks

Raul Martin[1], Blanca Lopez [1,2,*], Ivan Vidal [1,*], Francisco Valera [1], Borja Nogales [1]
[1]Telematic Engineering Department, Universidad Carlos III de Madrid, Avda. Universidad, 30, 28911 Leganés, Madrid, Spain
[2]IMDEA Networks Institute, Avda. del Mar Mediterráneo, 22, 28918 Leganés, Madrid, Spain
[*]Correspondence: blanca.lopez@imdea.org (B.L.); ividal@it.uc3m.es (I.V.)
https://doi.org/10.3390/app14031018

ABSTRACT

Quantum technologies promise major advances in different areas. From computation to sensing or telecommunications, quantum implementations could bring significant improvements to these fields, arousing the interest of researchers, companies, and governments. In particular, the deployment of Quantum Key Distribution (QKD) networks, which enable the secure dissemination of cryptographic keys to remote application entities following Quantum Mechanics Principles, appears to be one of the most attractive and relevant use cases. Quantum devices and equipment are still in a development phase, making their availability low and their price high, hindering the deployment of physical QKD networks and, therefore, the research and experimentation activities related to this field. In this context, this paper focuses on providing research stakeholders with an open-access testbed where it is feasible to emulate the deployment of QKD networks, thus enabling the execution of experiments and trials, where even potential network attacks can be analyzed, without the quantum physical equipment requirement, nor compromising the integrity of an already built QKD network. The designed solution allows users to automatically deploy, configure, and run a digital twin environment of a QKD network, offering cost-effectiveness and great flexibility in the study of the integration of quantum communications in the current network infrastructures. This solution is aligned with the European Telecommunications Standard Institute (ETSI) standardized application interface for QKD, and is built upon open-source technologies. The feasibility of this solution has been validated throughout several functional trials carried out in the 5G Telefónica Open Network Innovation Centre (5TONIC), verifying the service performance in terms of speed and discarded qubits when generating the quantum keys.

1 INTRODUCTION

Since the development of the Quantum Theory, applied physicists and engineers have been thinking about different ways to apply the unique features quantum objects present to design new technologies and useful products. Nowadays, it is common, for example, to find Non-deterministic Random Bit Generators (NRBG) in modern electronic products. These generators use a quantum process as an entropy source to generate random bits for critical applications, like random key generation.

Even more disruptive technologies are just around the corner. From engineering and design to biology or medicine, quantum computing will bring significant advances in all these fields [1, 2]. However, this

powerful machinery poses a threat to the security of all networks. Currently, the information we exchange is encrypted using keys formed according to algorithms, such as the RivestShamirAdleman (RSA) [3] or the DiffieHellman (DH) [4] algorithms, which are hard to break using an ordinary computer. But, since the late 1990s, mathematicians have been proposing quantum computing algorithms, like the paradigmatic case of Shor's algorithm [5], which would break the vast majority of these classic cryptographic algorithms without too much hassle. Since quantum computers capable of implementing those quantum algorithms are becoming a reality, anyone with a quantum computer powerful enough would be able to decipher all the classically protected information exchanged by two network endpoints.

How to protect sensitive information against this new technology is not something that has gone unnoticed by the scientific community. In fact, one of the most developed applications of quantum technologies is Quantum Key Distribution (QKD), where the security level of the communication between two nodes is based on general principles of Quantum Physics and, therefore, it does not depend on any computational assumptions. If the day when Quantum Computation is widely available comes, quantum networks protected by QKD will be an enabling technology to ensure the security of the information exchanged.

The first prototypes of QKD networks are already being built around the world [6–8], and thanks to advances in detectors, transmitters, and measurement protocols, results are being achieved that break previous QKD speed and distance records [9–11]. However, quantum devices and equipment are still under development, making their availability low and their price high, hindering the deployment of physical QKD networks, and thus slowing down the process of protecting the information exchanged in our networks from new and powerful attacks.

Given that QKD is a key technology to be developed in an era of paradigm shifts and that not everyone has current access to a physical QKD network, or cannot risk the network for the sake of experimentation, this paper presents a solution that automatically deploys a digital twin of a QKD network with any desired topology. The solution offers a service that allows automatic build and distribution of QKD nodes in virtual machines or physical devices, thanks to the implementation of Network Functions Virtualization (NFV) technologies. We use a quantum channel emulator which can be executed in these QKD nodes. With this, we can flexibly deploy emulated QKD nodes at different locations, a feature that, to the authors knowledge, is not available in existing state-of-the-art quantum emulation tools. This service relies solely on open-source technologies and is publicly available in [12] to ensure that any researcher interested in this field can carry out experiments and tests in order to develop QKD network technology without the requisite of having access to, or compromising, the machinery of a physical QKD network. The presented solution has been validated with several tests conducted in the 5G Telefónica Open Network Innovation Centre (5TONIC), an advanced research and innovation laboratory founded by Telefónica and IMDEA Networks. These tests show that the service can deploy, configure, and run an 8-node QKD network in ap-

proximately 7 min, demonstrating the suitability of the platform to agilely perform research and experimentation activities on moderately complex QKD environments without incurring the capital and operational expenditures of a physical QKD ecosystem.

This document is structured as follows. Section 2 gives some background on the context of the developed service, reviewing basic concepts about QKD and describing the state-of-the-art in topics related to the service presented. In particular, it discusses digital twins, the difference between bits and qubits, reviews the main ideas of QKD and some of the standards already published on QKD networks, and gives a brief introduction to NFV technologies. Later, the design, functioning, and capabilities of the service are explained in Section 3. Technical and precise details about the software implementation are provided in Section 4. Section 5 presents the validation and performance evaluation results of the service. Lastly, Section 6 outlines the conclusions and future lines of our work.

2 BACKGROUND

Simulated and virtualized models have been proving their worth for decades. NASA has been simulating its space machinery for more than 50 years, successfully finding solutions through the simulation of different scenarios, for example, to the explosion of an oxygen tank on Apollo 13 shortly after its launch. This kind of technology is considered to be the precursor of digital twins.

A digital twin is a virtual, (i.e., softwarized) replica of a physical object, which accurately models its complex behavior without necessarily having to interact with it. It may be used to resemble a physical entity, enabling testing and experimentation with technologies of interest without the requisite of having access to, or compromising, the machinery of the physical entity. This is the same digital twin approach followed in [13, 14], for example. Having a virtual copy of the subject matter offers many advantages, such as allowing the simulation of several scenarios without having to physically build them, providing major flexibility and cost-effectiveness, or assisting in the prediction of problems of its physical twin, which allows for staying one step ahead in resolving errors in the physical entity [15, 16].

Nowadays, digital twin technology can be found in many different fields: from manufacturing or aviation to retail, healthcare, or smart homes and cities [16, 17]. In the networking field, digital twin technology also offers interesting benefits that are now beginning to be exploited. For instance, modern networks are constantly growing either in complexity, size, number of connections, or traffic, making them difficult to support. Using digital twins, and leveraging NFV orchestration tools, the performance of large infrastructures can be estimated quickly and accurately, allowing the anticipation of when an existing network will run out of resources based on a given growth [14].

Since QKD networks are currently not easy to build, but are a particularly attractive technological element to develop, a QKD digital twin environment may offer a powerful solution, enabling experimentation in this field without the need to have access to real quantum infrastructures. Thus, the features discussed

above, including flexibility and scalability, open a door to interesting and novel studies such as the integration of quantum communications in the current network infrastructures, as it can be potentially incorporated into larger experiments on this issue.

Building a digital twin requires a deep knowledge of the physical object to virtualize. Here, some basic concepts about QKD, along with the state-of-the-art of quantum networks are reviewed. Also, a look at quantum network simulators is taken, as it will be an essential part of virtualizing the QKD network. Lastly, a brief outline of NFV orchestration is stated.

2.1 Bits and Qubits

Unlike classical communication, where information is presented in the form of binary units called bits, which can be either in state 0 or 1, quantum information relies on quantum bits or *qubits*.

Qubits, as well as every quantum object, exhibit characteristics that have no equivalent in the classical world. The state of a qubit can be a combination (superposition) of the two classical states of a bit, 0 and 1. Moreover, the mere observation (measurement) of the state of a qubit disturbs it irreparably. Therefore, if a qubit is in a superposition state and is measured, it will *collapse* in one of the *measurement basis* states, with probabilities that depend on the original state. After the measurement, the initial state of the qubit is destroyed. Note that, if the initial state of the qubit corresponds to one of the measurement basis states, the result of the measurement can only be that state. The standard basis for qubits is the one whose states are the classical states of the bits, 0 and 1, but there is an infinite number of measurement bases that can be chosen depending on the situation.

Another relevant characteristic of qubits is that, unlike classical bits, it is not possible to copy an unknown qubit state while keeping the original intact, as the no-cloning theorem states [18].

Last but not least, qubits also experience the phenomena of entanglement. Entanglement is a unique quantum feature that implies the existence of global states of composite systems which cannot be written as a product of the states of individual subsystems [19]. This means that two entangled qubits cannot be treated as individual entities: whatever happens to one of them will inevitably have repercussions on the other. For example, if one of the qubits of the entangled pair is measured, it will break the entanglement and the state of the other will collapse as well, no matter how far apart they are, or how much time has passed since the start of the entanglement. There are ways to test the entanglement between qubits. For example, a CHSH test can be performed [20].

2.2 Basics of QKD

QKD is a cryptography scheme that allows the exchange of secret keys exploiting the quantum features of the qubits, basing its security solely on general principles of Quantum Physics, and not on computational assumptions like most current cryptography schemes.

The measurement principle makes it virtually impossible for an unwanted party to intercept the infor-

mation encoded in a sequence of qubits without being detected, since to obtain the information it will have to measure the qubits, thus destroying the original states. To understand the implications of the no-cloning theorem let us think on how easy it is to copy a sequence of bits: anyone who can intercept the bits and read them can produce an exact copy of that sequence and forward one of the copies to the genuine receiver (a man-in-the-middle attack). In contrast, the no-cloning theorem guarantees that copies of the state of a qubit cannot be created without destroying the original state [18]. Entanglement also protects the information encoded in qubits against eavesdroppers, since when one of the qubits of the entangled pair is measured, the entanglement is broken and, as we have discussed above, there are ways to test the entanglement between qubits.

Many QKD protocols, which describe the following steps to share a quantum key securely, have been proposed since the 1980s [21]. For historical reasons, a review of two of the first proposed protocols is presented.

- **BB84 protocol:** it was the first developed QKD algorithm, proposed by Charles Bennet and Gilles Brassard in 1984 [22]. In BB84, two quantum nodes, let us say, Alice and Bob, exchange a key following some steps. First, Alice encodes each bit of the key on the state of a qubit, randomly choosing between two bases in which she prepares the qubit and then sends it to Bob. When Bob receives the qubits, he measures them in one of the two bases Alice used, obtaining a deterministic measure if the basis is the same as the qubit was prepared in, and a probabilistic measure if he chooses the other basis. Then, they share classical information about the bases they used and only keep those measurements where their choice of basis was the same, i.e., those in which they are sure to have obtained the same values. Additionally, they perform an error detection check by sharing a subset of the final bits to identify possible eavesdroppers.
- **E91 protocol:** being proposed by Artur Ekert in 1991 [23], the E91 protocol was the first algorithm to exploit the phenomena of quantum entanglement. In this algorithm, both nodes exchange Bell pairs, i.e., maximally entangled states, and measure them on a randomly selected basis among a set of three bases. The basis sets of Alice and Bob differ on one basis, and they are specifically chosen to allow them to perform a CHSH test and verify if an unwanted party was eavesdropping on the communication (note that if the test is not passed, someone might be measuring the qubits and, therefore, breaking the entanglement). After the measurement of all the entangled pairs, they share the basis used to measure each qubit and keep only the bits from the measurements where the same basis was used, which again implies that the measurement will be the same in both nodes. Around $7/9$ of the exchanged qubits are typically discarded due to the construction of the algorithm itself: there are 9 possible basis combinations and only in 2 of them do Alice and Bob choose the same basis. The discarded qubits are used to perform a CHSH test.

2.3 QKD Networks

Although QKD networks, and quantum networks in general, are still in an early development phase due to the relatively new appearance of equipment that supports them, they are expected to be an important advance that significant organizations, such as European Telecommunications Standard Institute (ETSI) or Internet Engineering Task Force (IETF), have created specific committees [24, 25] to develop standards that can serve as a framework to build future quantum networks. Several documents of this nature have already been published in the last years, like the Architectural Principles for a Quantum Internet RFC 9340 [26] by the IETF. These documents also include specifications on Application Programming Interfaces (APIs) for QKD nodes [27, 28], use cases for the Quantum Internet [29], or descriptions of the main communication resources involved in QKD systems [30].

Apart from these standards, there are also several research lines where quantum network layered architecture is being developed. For example, in [31], the authors explore how QKD networks could be deployed in next-generation infrastructures, proposing an architecture for a Software-Defined QKD Network Node. There are also several surveys on QKD networks deploying techniques and practical implementation, which provide reviews of the different layered descriptions of QKD networks used [32, 33]. However, there is still not a widely accepted model for a quantum network architecture.

A description of the main components of a QKD network node can be found in the ETSI GS QKD 004 Quantum Key Distribution (QKD) Application Interface document [27], along with an example of two sites in a QKD network, shown in Figure 1.

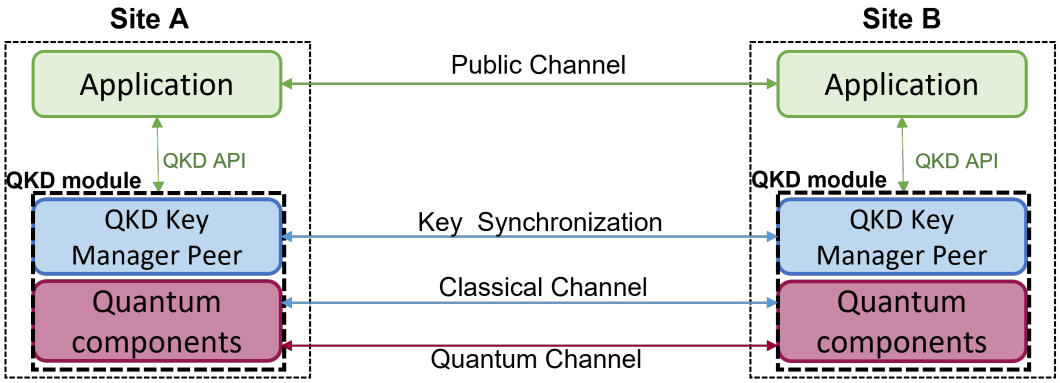


Figure 1: General example of two sites in a QKD network by ETSI [27].

In the scheme that can be seen in Figure 1, there is an upper layer corresponding to the client applications. Underneath, there is a component named a QKD module, marked with a dashed line in the same figure, which can communicate with the application layer through a QKD API defined in the same document [27]. The QKD module of each site is formed by a QKD Key Manager Peer in charge of managing the keys produced in the lower layer, the Quantum components layer of the node. This last layer is where the QKD protocol for forming quantum keys is implemented.

2.4 Quantum Network Simulators

A fundamental component of the service will be the quantum network emulator used to build the quantum channels in the virtual QKD network. There is a variety of quantum network simulators and emulators that allow simulating the transmission and operation of qubits between two nodes [34]. The choice of the simulator or emulator to be used is a fundamental part of the development of the service, as some features of the QKD network digital twin will depend on which software solution is used.

One of our design criteria was to build the service as open-source software, making it available to the research community, network engineers, and stakeholders working in the field of quantum networks and QKD. Flexibility was also a fundamental requirement, that is, the service must be able to deploy digital twins of moderately complex QKD networks, built by multiple QKD nodes and quantum links. Moreover, deployments should be realized within a reduced time frame, requiring minimal intervention from the user, who should only provide the details of the QKD network to be deployed as a digital twin. The digital twin must in turn resemble as much as possible the real QKD network, and as no real network runs locally on a single device, a simulator that offers the possibility of distributing the network nodes among different machines is needed. To ease the implementation or elimination of different functionalities, a modular design is sought, which allows working on different components of the digital twin independently.

In the following, we present a brief review of the quantum simulators and emulators that have been studied for the development of the service:

- **SeQueNCe** is a photonic network simulator implemented in Python, where a metropolitan-scale quantum network was successfully modelled using single photon simulations [35]. It is divided into five modules, from the hardware to the application layer, with cross-module communication. This simulator allows the modification of many classical and quantum parameters, from the attenuation and the delay introduced by the channels to the fidelity of entanglement between two qubits, or the efficiency of the single-photon detectors. It also includes an implementation of quantum memories and several entanglement-related protocols [35].
- **QuNetSim** is a Python simulator designed to model the network layer and above instead of focusing on the simulation of the physical properties of quantum networks [36]. It includes many existing basic protocols as a toolbox to make easier the development of more complex protocols. It can process both classical and quantum information and uses a network layering model based on the Internet architectural model, with three layers: application, transport, and network [36].
- **NetSquid** is a modular simulator available as a Python package. It allows modelling different physical quantum devices that can appear in a quantum network, as quantum memories [37]. Users can

choose between a detailed description of the quantum states using wave functions or density matrices, or a simplified version using the stabiliser formalism. It can support the simulation of large-scale networks thanks to its computation engine, which can manage multiple qubits simultaneously [37].

- **The Parallel Simulation Framework** is an optimized parallel simulator, proved to be nine times faster than a sequential simulator in an experiment involving a 64-node QKD network [38]. It has two different layers, the upper one corresponds to the modelled hardware and protocols, and the lower layer consists of the kernel responsible for the parallel discrete event simulation. It only implements the BB84 protocol but can be expanded to other schemes [38].
- **SimulaQron** is a Python emulator with a modular design, that allows running the different quantum network nodes on different physical or virtual machines. By emulating quantum and classical channels, it enables the execution of quantum applications between remote devices, managing the operations and state of the qubits [39]. It implements its own protocol, the CQC protocol [39], to achieve the communication between applications and physical quantum equipment (either real or emulated by SimulaQron itself). Although it has been mainly developed in Python, it can be programmed using any language capable of connecting to the CQC server backend using a TCP connection and sending packets in the required form [39].

Table 1 presents a comparison of the considered quantum network simulators and emulators. Each column takes into account one of the sought features discussed previously: the possibility of distributing the nodes of the QKD network in order to resemble a real network as much as possible, and the modularity to facilitate the independent development of features in different components of the network. The last column, called “Simulation environment”, refers to the programming language used to write the quantum applications that can be run on each solution. N/A in this last column means that the authors did not specify the programming language of their tool.

Table 1: Comparison of the different quantum network simulators considered [34].

Simulator	Publicly Available	Distributed Nodes	Modularity	Simulation Environment
SeQueNCe	Yes (open source)	Not allowed	High	Python
QuNetSim	Yes (open source)	Not allowed	Low	Python
NetSquid	Yes (upon registration)	Not allowed	High	Python
Parallel Simulation Framework	No	Not allowed	Low	N/A
SimulaQron	Yes (open source)	Allowed	High	Python

Although each of them has unique strengths and constraints, the publicly available emulator SimulaQron was chosen for our implementation since, to the authors' knowledge, it is the only one that allows implementing a quantum application on separate network nodes, emulating realistic quantum communications between these nodes. In addition, SimulaQron presents a modular design that eases implementation aspects and offers comprehensive Python implementation of quantum operations that facilitates the development of quantum applications.

2.5 Service Orchestration

The advent of the fifth generation of mobile networks, 5G, has brought profound advancements in delivering value-added services to end-users. Beyond defining a new radio access network architecture, aimed at enhancing the overall performance of end-user communications, 5G has introduced novel technological paradigms and business models to meet the stringent performance requirements demanded by 5G services. In particular, softwarization and virtualization [40] play a pivotal role in 5G, facilitating the swift deployment of complex services as a composition of individual software components. These may be executed in virtual machines or lightweight virtualization containers [41], which may strategically be deployed close to end users leveraging cloud and edge infrastructures of service providers. This way, service delivery can be enhanced from an end-user perspective, e.g., in terms of perceived latency and content transmission rates, while reducing capital and operating expenditures. Still, it is important to note that this approach for service delivery requires specific control procedures, which are able to manage and orchestrate the deployment and the proper configuration of the software components of every service at their respective locations.

Service management and orchestration has received intense attention from the research community and standards-developing organizations. In this respect, the ETSI has led the definition of the NFV technology [42]. ETSI NFV provides a reference framework to support the automated deployment of telecommunication and industry-specific services on cloud and edge facilities, referred to as NFV infrastructures. These research and standardization efforts have resulted in numerous open-source solutions that have been used in different contexts to provide an operational implementation of the ETSI NFV reference architecture. Well-known examples of these solutions are Open Source MANO (OSM) [43], Cloudify [44], and OpenStack [45]. More recently, the open-source community has started to consider the integration of cloud-native solutions, such as Kubernetes [46], into the ETSI NFV framework. The cloud-native paradigm enables novel approaches for service development and deployment, decomposing a service into smaller units that inter-operate, known as microservices [47], and executing microservices in portable, lightweight virtualization containers.

The design of our solution is inspired on ETSI NFV management and orchestration principles. The user describes a QKD network as an NFV service, that is, as a composition of interconnected QKD nodes. Our digital twin solution incorporates, as will be covered in the following sections, state-of-the-art technolo-

gies and open-source NFV tools, in order to support the automatic deployments of QKD network digital twins.

3 FUNCTIONAL OVERVIEW

One of the main properties we pursue in the development of the QKD network digital twin service is that the architecture of the digital twin environment follows current standards. Specifically, we focus on those that present a general definition of QKD node components and define an API between applications and the QKD module of a node [27, 28]. In addition, we require the ability to distribute the network nodes among different machines, as discussed in the Section 2.4, and we also look for the convenience of automatic deployment of the desired QKD network, allowing the use of an existing NFV orchestration tool.

In our service, for each QKD node, the user may indicate specific configuration parameters, including the ETSI APIs and the QKD protocol that must be supported. With this, our solution provides a management and orchestration service, handling the complete lifecycle (creation, configuration, and termination) of a digital twin of the QKD network. The digital twin implements each QKD node in the software, enabling its execution in a virtual machine or a virtualization container. We use an emulator to support quantum links between QKD nodes and qubit operations, such as creating and exchanging Bell pairs. The management and orchestration service handles the deployment of the digital twin of the QKD network, as a composition of virtualized QKD nodes. Moreover, QKD nodes may be placed at different locations at the discretion of the user. To this purpose, our solution can leverage ETSI NFV-compliant cloud and edge infrastructures, (e.g., based on OpenStack), if available. Alternatively, it may set up the digital twin of the QKD network using physical or virtual machines, or virtualization containers pre-provisioned by the user. After the deployment process, user applications can access the different QKD nodes to retrieve cryptographic key material, using their respective ETSI standard interfaces.

As can be deduced from the above discussion, in the developed QKD network digital twins service, there are three important roles: the digital twin orchestrator, the QKD network nodes, and the QKD network clients.

3.1 QKD Network Digital Twin Orchestrator

The QKD network digital twin orchestrator is responsible for deploying, configuring, and terminating any desired QKD network digital twin defined by the user of the service. As can be seen in Figure 2, a total of three steps must be completed to achieve this.

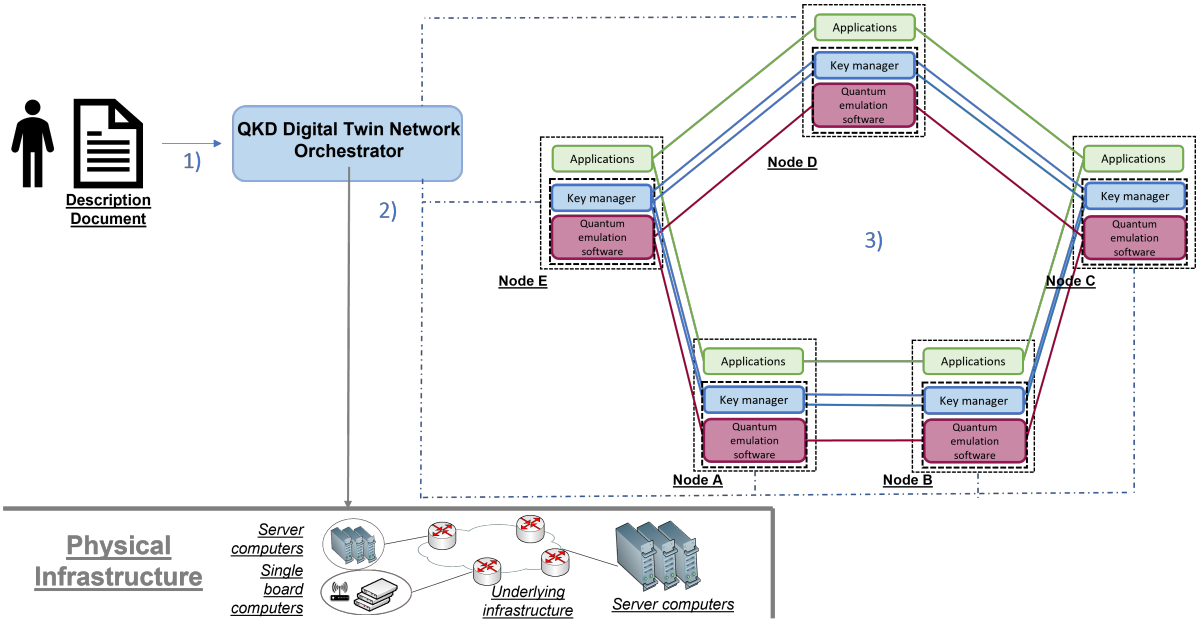


Figure 2: Outline of how the QKD network digital twin orchestrator service works. The numbers indicate the order in which the steps are carried out.

1. The service user must describe the desired QKD network and deliver this description to the orchestrator. In particular, this document has to include parameters such as the name and neighbors of each node, the ETSI API to be used, or the QKD protocol emulated when forming the quantum keys.

2. Once the digital twin orchestrator has the QKD network description, it communicates with the available physical infrastructure to instantiate the nodes comprising the digital twin of the desired QKD network. At this point, there are two possible scenarios. In the first case, the user provides the orchestrator with a set of pre-deployed machines (real, virtual, or virtualization containers) where he wants the digital twin of the QKD network to be deployed. If this is not the case, the ETSI NFV-compliant cloud and edge infrastructures (e.g., OpenStack-based) must be accessible to the orchestrator for it to automatically build the digital twin of the QKD network using standardized mechanisms to request the deployment of the virtual machines or virtualization containers needed to run the different QKD network nodes.

3. Lastly, the orchestrator installs the required software in order to emulate the complete behavior of the QKD nodes, including that which enables communication based on the ETSI API chosen by the user. In this last step, the orchestrator configures the digital twin and initializes the software that emulates the quantum channels and the QKD protocol used to form the quantum keys.

Once these processes have been successfully completed, the QKD network starts operating and is available for any client application.

3.2 QKD Network Nodes

The main components of a QKD network are the QKD nodes. To achieve the correct functioning of all QKD network features, the architecture and components of the QKD nodes must be the appropriate ones.

As discussed in Section 2.3, there is still not a widespread layered architecture for QKD networks as in classical communications; however, there are several research lines where this is being developed [31–33]. The most common element in the proposed architectures is a Key Manager, an element that is also present in classical key distribution systems. To guide our service, we assume a basic functional architecture, built upon the QKD network site scheme from ETSI [27], thus aligning the service with current standards. The functional architecture of the QKD network digital twin sites can be seen in Figure 3. The same colour code as in Figure 1 is followed, which helps to compare the standard model with our design.

The top layer of a site, as in the ETSI scheme, corresponds to the application layer. It can communicate with the matching layer in other network sites by classical processes, and with the QKD module of its site through the QKD network client component where the QKD API is implemented. Below this top layer is located what we call the QKD module of the site, similar to the ETSI scheme, where the QKD API is also implemented, enabling applications to communicate with the module. The QKD API has been developed following the ETSI standardized API described in [27].

Within the QKD module, there are three layers that correspond to the QKD Key Manager Peer component in the ETSI scheme. The upper layer, the Key Management layer itself, is in charge of managing the different key streams and delivering the keys to the corresponding applications. Then, the Key Synchronization layer, ensures the keys for each key stream are exchanged in a synchronized way in all sites. Lastly, the Quantum Backend layer is responsible for communicating with the actual quantum equipment to trigger different quantum operations (e.g., creating a local qubit, creating a Bell pair shared with another node, or applying an operation to a qubit). The bottom layer of the QKD module corresponds to the quantum components. In our service, the quantum behavior and the creation of quantum keys are emulated through software.

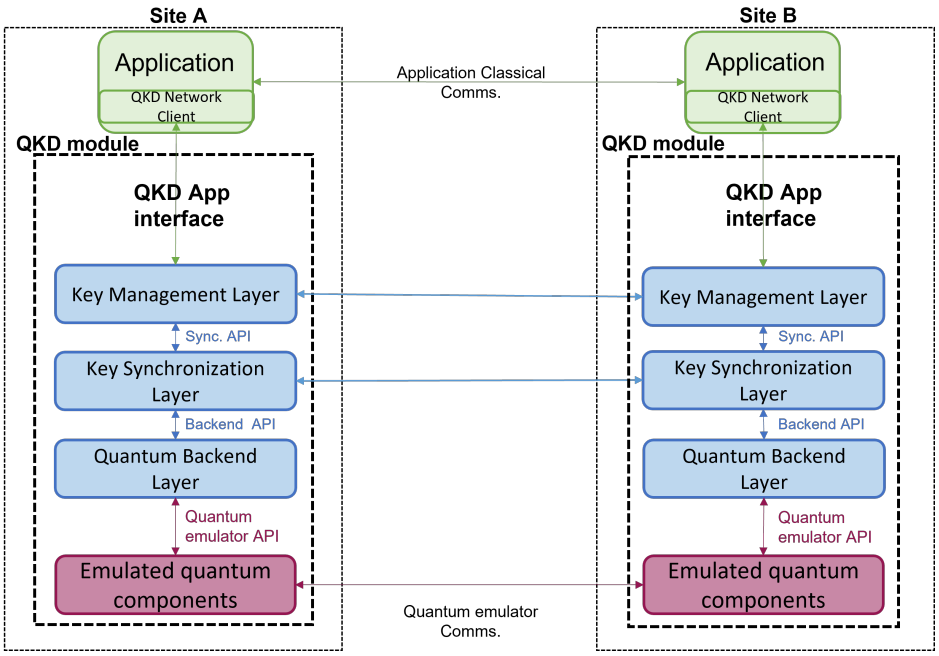


Figure 3: Functional architecture of the QKD network digital twin sites. Components along with their interfaces are shown.

Components at the layers can vertically interact with components at other layers immediately above and below them, through specifically developed APIs. They can also communicate horizontally with components in the corresponding layers of other network sites. Note that although in the scheme proposed in the ETSI standard [27], shown in Figure 1, the quantum components of different sites are linked by two kinds of channel, our design implements both using a quantum emulator which is capable of emulating quantum and classical channels.

3.3 QKD Networks Clients

The clients of a network are the requesting party, in our case, the applications that request keys from the QKD modules. A real example of these applications could be routers that need to settle an encryption key or a malicious application that tries to obtain that encryption key. The digital twin deployed by our service enables the testing of these applications over a realistic QKD network environment, supporting research and experimentation in this field.

In the presented service, any device with access to the digital twin of the QKD network can potentially be a client and run applications over the QKD network environment. This includes both external and internal QKD network devices. In order to make their requests and receive their responses, clients must use a specific API. In the first version of this service, the API corresponds to an implementation specifically designed based on the one defined by ETSI GS QKD 004 [27].

3.4 Service Capabilities

In addressing shortcomings in the access and implementation of secure QKD networks, we have developed a digital twin service that offers significant advantages. This service allows users to construct virtual replicas of such networks by simply providing a description of the desired deployment. Additionally, it provides a working environment aligned with a standardized API, meaning that the emulated network adheres to the same guidelines as a real QKD network.

Users can address QKD-related issues without the need to access the physical infrastructure of a real QKD network, resulting in significant savings of resources, both in terms of time and funds, which would otherwise be required to build a real network. Leveraging virtualization and NFV technologies, our service can easily scale both horizontally (in terms of the number of QKD nodes) and vertically (in terms of the processing capacity of each QKD node). Notably, to the best of our knowledge, no other service allows for the automatic deployment of distributed emulated QKD nodes, as is the case with the service we are presenting.

Furthermore, the service also solves the typical problems of incompatibility of devices from different manufacturers, i.e., it offers much more flexibility in testing diverse network designs, as all elements communicate with the same ETSI API. The modular Python design we have employed facilitates implementation and integration of new developments, standards, protocols, and more.

Our approach not only enables the emulation of QKD nodes but also opens up multiple possibilities regarding its utilization. These include but are not limited to conducting research into securing novel Internet applications through QKD, testing new protocols and APIs for the development of standards in the QKD domain, experimenting with large multi-operator QKD networks in a realistic manner, and working on integrating independent islands of QKD nodes offered by different service providers.

4 IMPLEMENTATION

To achieve the full functioning of the roles discussed in Section 3, three main software components have been developed [12]: an orchestrator application to manage the life cycle of the QKD network digital twins, and two Python packages, one for the QKD network modules and other for the QKD network clients, where the QKD API functions are defined to ease the development of applications. This section gives an overview of the most relevant implementation details of each of them.

4.1 QKD Network Digital Twin Orchestrator Application

The QKD network digital twin orchestrator application allows the user to deploy, configure, and start a QKD network digital twin with any desired topology. The input of the application is two documents. The first, referred to as the configuration document, describes the desired QKD network in terms of its comprising nodes. Specifically, the document includes the names that identify each node and indicates how the nodes are connected (i.e., it includes the list of nodes, or neighbors, to which each node is connected).

In addition, the configuration document includes information about the standardized QKD API to be used (in the case of the first version it should be ETSI GS QKD 004 [27]), and the QKD protocol to form the quantum keys. The current implementation of the service supports a variant of the E91 protocol, as it represents a complex example of QKD protocol with quantum capabilities, such as entanglement management. The second document is the inventory document, and it serves to set relevant parameters to enable the configuration of the nodes from the orchestrator. These parameters include the IP addresses of the nodes, the runtime environment for the software on which the configuration of the nodes is based (i.e., Python 3 and pip, the package manager for Python), and the login credentials of the nodes to access them. Both documents are based on the YAML format, which is a type of file very common when using management and orchestration (MANO) solutions such as OSM or Kubernetes.

When deploying a digital twin of a QKD network using our service, two scenarios arise, where the user has or does not have the necessary machines pre-deployed.

On the one hand, we consider a scenario where the user has a pool of running machines on which to emulate the implementation of a QKD network. In this case, those machines must be accessible to the orchestrator of our service, and be compliant with the service requirements. In addition, the user must include the IP addresses and the credentials of the machines in the inventory file so that the orchestrator can connect to them. Then, the orchestrator will configure the QKD network digital twin, downloading the required software in the machines, and connecting through emulated quantum and classical channels the QKD nodes that the user had described as neighbors in the configuration documents.

On the other hand, if the user does not have a pre-deployed accessible machines pool, the orchestrator can deal with the deployment of the machines that will play the role of the nodes of the emulated QKD network. In this case, the orchestrator still needs an inventory file, although the IP address parameter will be completed by the orchestrator once the machines are deployed. When executing the orchestrator, along with the two YAML files, the user will need to include the *"osm"* execution parameter. This parameter allows the orchestrator to identify that it is necessary to carry out the deployment of the QKD network machines and that their deployment is intended to be done by using the ETSI NFV MANO solution (i.e., OSM). As commented in previous sections, our orchestrator is designed to be able to use compliant ETSI NFV infrastructures to instantiate the required virtual machines to build the desired QKD network digital twin. In this context, any Virtual Infrastructure Manager (VIM) supported by OSM (e.g., the widely used tool OpenStack) can be utilized. Although in this first version the QKD nodes are packed in virtual machines, since OSM also support solutions such as Kubernetes, future versions of the service will be able to make use of containers. To specify the intended VIM where we carry out the deployment, the orchestrator provides an additional execution parameter defined as *"vim_account"*. As well as in the case where the user has a pre-deployed machine pool, once the virtual machines are deployed, the orchestrator connects to them to configure and install the required software.

In any of both scenarios presented above, once the machines are ready, the orchestrator leverages Ansible (through the Ansible python API) [48] to configure and start the QKD network nodes. Ansible is a widespread tool in the field of task automation since it allows the execution of tasks in remote hosts without having to deal with the specific configuration details of each system. It should be noted that Ansible performs configuration tasks in parallel in all the machines, so that the machines are configured simultaneously. To this, Ansible assigns a configuration process to each of the machines to be configured. However, the default number of parallel processes Ansible supports is five [49]; which implies that if there are five machines, the configuration tasks will take the same time as if there were one; but if there are six machines, the task will take twice as long, since in that case two rounds of parallel tasks have to be executed. Using Ansible, the orchestrator will download the required software in the machines, including Python and the pip package if the machines were not pre-deployed, along with the QKD Node package that contains the quantum emulator SimulaQron, and that is capable of communicating with client applications through the developed QKD API. Then, the orchestrator will start the QKD network digital twin by connecting through emulated quantum and classical channels the QKD nodes that the user had described as neighbors in the configuration document.

From this point on, the QKD network environment is operational and applications can run over it.

Both scenarios described in this section can be found in Figure 4. On the left side of the figure, the scenario in which the user has a pre-deployed machine pool is shown. On the right side, there is the case in which the OSM is used to deploy the needed machines.

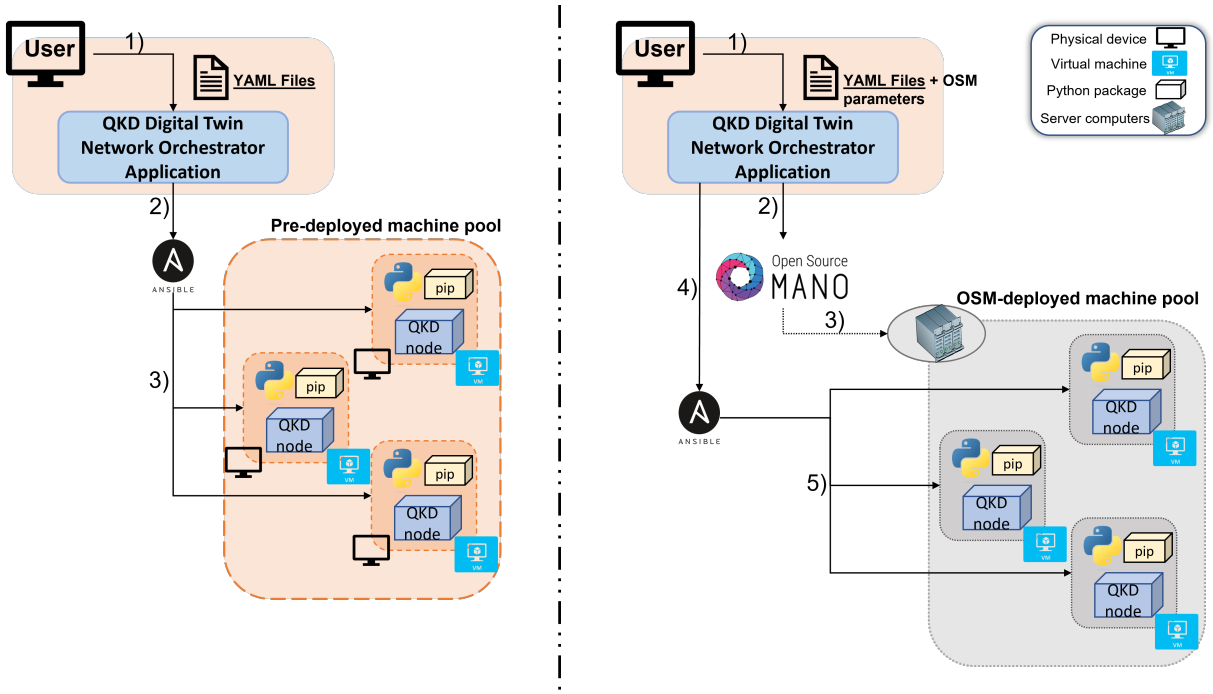


Figure 4: Left, steps followed by the QKD Network Digital Twin orchestrator application when a pre-deployed machines pool exists. Right, steps followed when the OSM is used to deploy the QKD network machines.

4.2 QKD Module Software

As discussed in Section 3.2, the implemented QKD network module is structured into different layers that communicate with one another via a specifically designed API. The upper layer of the QKD node and the client applications communicate with each other through the QKD API, designed and implemented for this purpose, and aligned with the ETSI API described in [27]. Within the QKD module, the other two specific APIs are implemented and used between the layers that form the QKD Key Manager Peer of different nodes. Lastly, the bottom layer emulates the quantum components of the node that enable the creation of quantum keys and uses the protocol implemented in the emulator itself to communicate with the layer above it and with the matching layers in other nodes.

The Python package developed for the QKD network module follows the architecture shown in Figure 5 and is the main component of the system. It runs all the functional layers on the node described in Section 3.2.

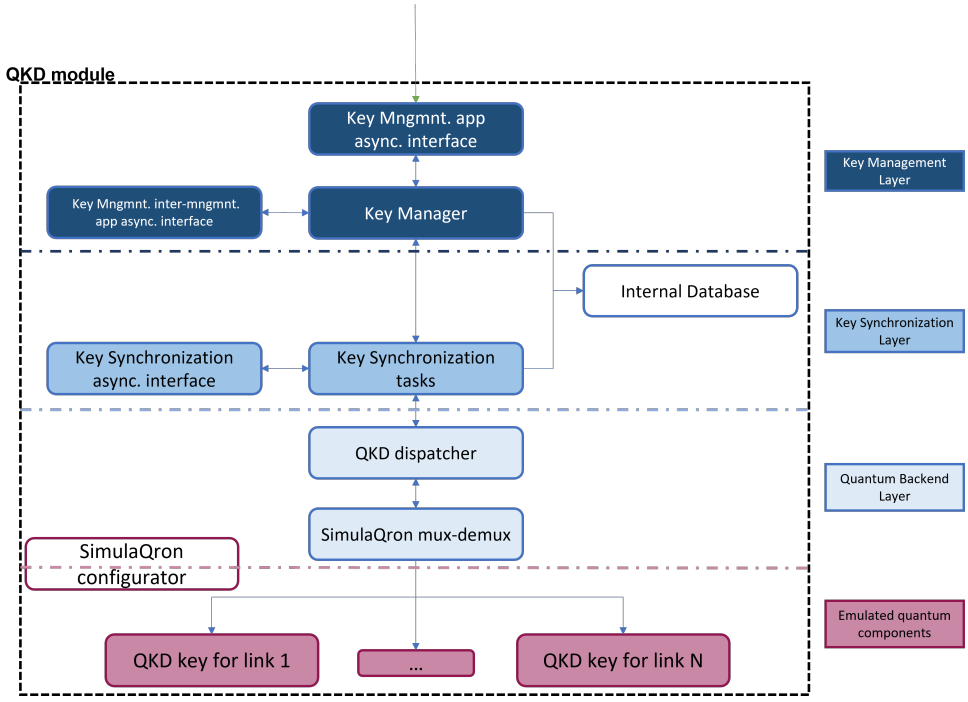


Figure 5: Overview of the software architecture of a quantum node. Layers can be identified by sections and colours.

From top to bottom, the QKD module package is structured as follows.

The upper layer of the QKD module, the Key Management layer, is formed by three software components. First, a Key Management application asynchronous interface is included, where the three functions defined by ETSI QKD 004 [27] are implemented using gRPC [50], a modern open source solution used to connect multiple services in an environment. This is the element that enables the communication between the QKD module and the client applications. The implemented API enables three methods of execution, corresponding to the three functions defined in [27]:

- ***open_connect***: This instruction indicates that a new stream of QKD-generated keys must be created with another node. To call this method, information about the applications that will access nodes involved in the new stream and other parameters, such as the lifetime of the keys, must be included. Once this method has been executed, a Key Stream ID (KSID), which can be used by the application to identify the key stream in subsequent calls to other methods, is generated. After calling this method on both nodes, the nodes start to continuously exchange and buffer keys for this key stream.
- ***get_key***: This method is used to retrieve a key from a particular key stream identified by its KSID. If no new key is available yet, an error code is returned. This method additionally allows the user to optionally send an "index" parameter to specify the key in the key stream it wants to retrieve for synchronization purposes.
- ***close***: This instruction closes and frees a particular key stream identified by its KSID.

Beware this is just a quick and simplified overview of the interface. For the complete picture and details, please refer to the specification document [27].

Below this interface, the Key Manager itself is found. It is responsible for managing key flows, i.e., it is the element that receives requests from applications and performs actions based on these requests, including the delivery of the requested keys to the applications. Lastly, a component named the Key Management inter-management application asynchronous interface is added to enable communication between Key Managers of different sites.

The next layer in the designed architecture of the QKD module is the Key Synchronization layer. This layer counts with a principal component named Key Synchronization tasks that ensures, while it may appear redundant, the proper synchronization in the key formation and delivery processes. This can be achieved because the quantum emulator assigns an identifier to each qubit, allowing it to pinpoint precisely each of them and easing the synchronization inside the QKD module. The other component of the Key Synchronization layer is the Key Synchronization asynchronous interface which, as well as in the case of the Key Manager layer, enables the horizontal communication between Key Synchronization layers of different sites, a crucial feature in order to settle cryptographic quantum keys between two sites of a network.

Underneath the Key Synchronization layer, the Quantum Backend layer is formed by two other components. The first one, the QKD dispatcher, is responsible for choosing the thread in which the QKD protocol for a specific key will run. This means that the QKD module can simultaneously create several quantum keys, depending on the characteristics of the machine on which the module is located. Below the QKD dispatcher is an element called SimulaQron mux-demux that adds a layer between the quantum emulator and the other components to perform a little information processing and send it properly to the quantum emulator. It also helps to manage the various threads produced by the QKD dispatcher.

Lastly, the emulated quantum component layer has as many components as there are QKD links on the site. For each QKD link, we use the quantum emulator SimulaQron to create the required keys following a QKD protocol. In the first version of the service, a variant of the E91 protocol is implemented to form the quantum keys. It is worth noting that SimulaQron allows emulating quantum and classical channels, necessary to complete QKD protocols between different machines, as well as performing local operations on qubits and managing the entanglement of qubits in different machines. This allows the creation of distributed QKD modules, making the digital twin environment much more similar to a real QKD network.

Two other components, which do not correspond to a specific layer, are also implemented. The first one is an Internal Database, which stores information related to the key streams and their buffered keys, such as the destination and the source of the streams, the KSID, or the sizes of the keys. It also notifies other components when a free space is left in the buffer. Both the Key Manager layer and the Key Synchronization layer have access to this database to correctly perform their corresponding tasks. The last component implemented in the QKD module is a SimulaQron configurator, which uses the SimulaQron API to transparently create a QKD network from the provisioning file and start tracking its execution.

All these components that form the QKD modules have been implemented in a single Python package that can be freely installed. In principle, the user of the service does not need to worry about the functioning of this package, since to write his or her applications and perform experiments over the QKD network digital twin environment, a QKD network client package has been developed specifically for this purpose, with the aim of making the service as accessible as possible.

4.3 QKD Network Client

In order to write their applications and deliver them to the QKD network digital twin, users must use a specific API, in the first version of the service, the one defined by ETSI GS QKD 004 standard [27].

To facilitate the creation of applications and to make it as user friendly as possible to conduct experiments on our QKD network environment, a QKD Network client package has been developed. In this package, the three executing methods that the implemented API supports have been defined; therefore, to write a Python application that makes use of the QKD network digital twin, the user may install and import this package, thus being able to easily call the three execution methods discussed in the previous section: ***open_connect***, ***get_key***, and ***close***.

5 VALIDATION AND RESULTS

Three different kinds of experiments have been conducted to validate the developed solution. All tests have been carried out in the 5G Telefónica Open Network Innovation Centre (5TONIC) [51], which counts with an NFV infrastructure based in OpenStack and OSM [52].

In order to test the two possible uses of the QKD network digital twin orchestrator discussed in Section

4.1, we have conducted two of the experiments over a pre-deployed machine pool, while the third one was performed using our service OSM feature.

5.1 Scenario Preparation

To provide the reader with an example of how the service is used, a template of both the configuration and inventory YAML files introduced in Section 4.1 and employed in the experiments can be found below.

In the configuration template that can be seen in Figure 6, the user must specify the version of the service to use (currently there is only one version, the 0.1.0 indicated in the YAML template), the API, and the QKD protocol. Then, the nodes have to be named, indicating their IP addresses and their neighboring nodes. Note that the IP address parameter can be left blank or filled with any value in the case of using the OSM feature of the service.

```yaml
qdts_version: 0.1.0
config:
  application_interface: etsi-gs-qkd-004
  qkd_protocol: e91
nodes:
  - node_name: Alice
    node_ip: 10.4.16.115
    neighbor_nodes:
      - Bob
      - Charlie
  - node_name: Bob
    node_ip: 10.4.16.74
    neighbor_nodes:
      - Alice
      - Charlie
  - node_name: Charlie
    node_ip: 10.4.16.132
    neighbor_nodes:
      - Alice
      - Bob
```

Figure 6: YAML configuration file describing the scenario.

The inventory document, of which a template can be found in Figure 7, is written in the form of an Ansi-

ble inventory. It collects the necessary parameters for Ansible to connect to the nodes, providing it with the IP address of the node, the target user, and its password. The last parameter named “py_env” refers to the directory where Python is installed, needed to download the packages required for emulating the QKD network. Note again that if the user intends to use the OSM function, the IP address parameter, named in the inventory file as “ansible_host”, can be left blank.

```
all:
  hosts:
    Alice:
      ansible_host: 10.4.16.115
      ansible_connection: ssh
      ansible_user: ubuntu
      ansible_ssh_pass: ubuntu
      py_env: /usr/bin/
    Bob:
      ansible_host: 10.4.16.74
      ansible_connection: ssh
      ansible_user: ubuntu
      ansible_ssh_pass: ubuntu
      py_env: /usr/bin/
    Charlie:
      ansible_host: 10.4.16.132
      ansible_connection: ssh
      ansible_user: ubuntu
      ansible_ssh_pass: ubuntu
      py_env: /usr/bin/
```

Figure 7: YAML inventory file describing the scenario.

5.2 Functional Validation

The first experiment presented is a functional validation for the main functionalities of the developed solution, such as key formation and synchronization between nodes. The scenario used in this experiment consists of three pre-deployed machines over the OpenStack infrastructure in 5TONIC. Each of the virtual machines operates as a site in a QKD network. A scheme of this scenario can be found in Figure 8.

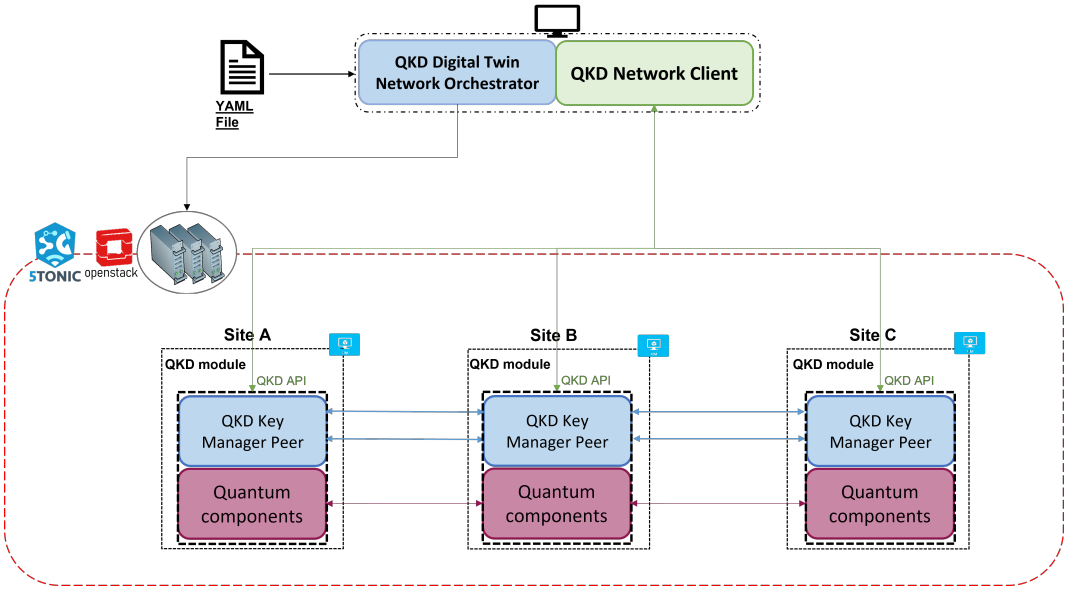


Figure 8: Scenario scheme of the functional validation and the tests regarding key exchange time and discarded qubits.

In this experiment, the QKD network client defines a single application using the QKD Network client package, thus communicating with the QKD network digital twin making use of a standardized API (the ETSI QKD 004 [27]), in the same way he or she would communicate with a real QKD network.

In each site of the scenario, the application creates key streams with the neighboring sites, retrieves the exchanged keys, and checks that they match for the connected nodes. The exact sequence of actions executed in the test is shown in Figure 9.

Following the scenario topology, which consists of three sites connected in sequence, two key streams are created: one between Module A and Module B, and one between Module B and Module C. Once 20 keys have been exchanged in each stream, the test application retrieves them at all the nodes and compares them by pairs, verifying that the keys of the neighboring nodes match.

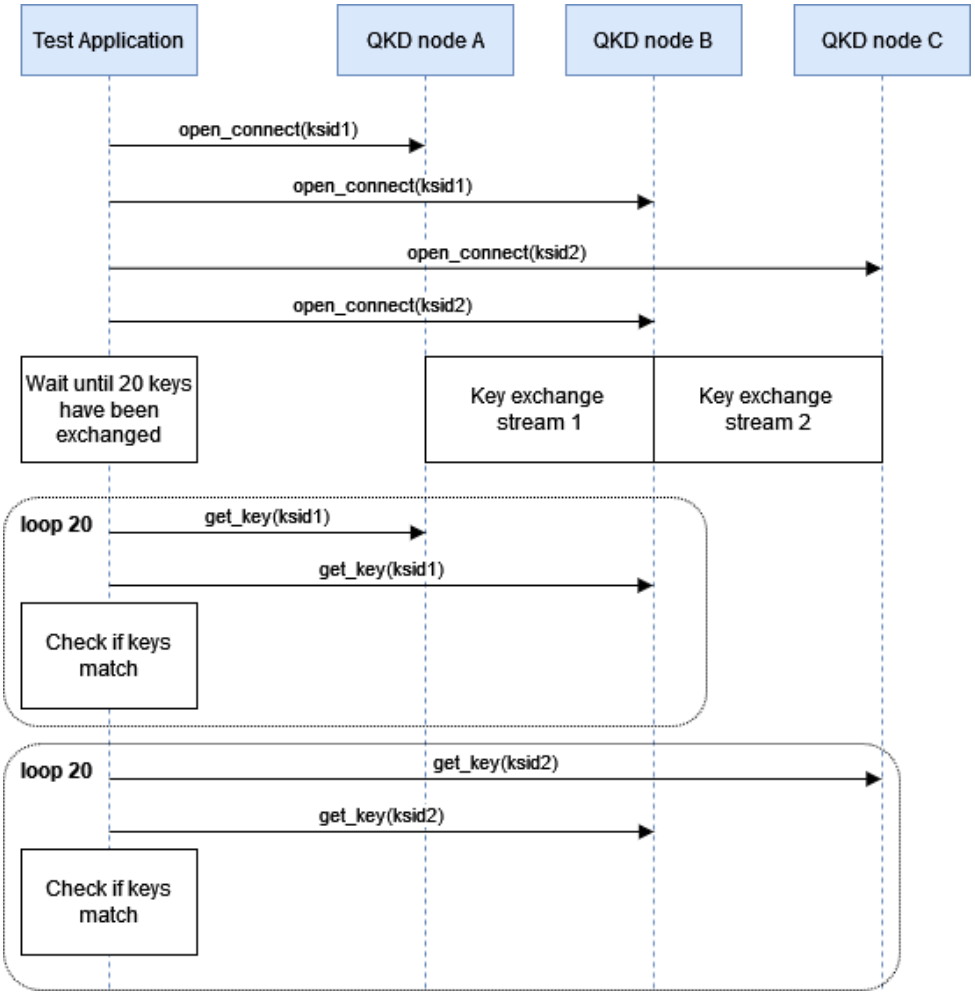


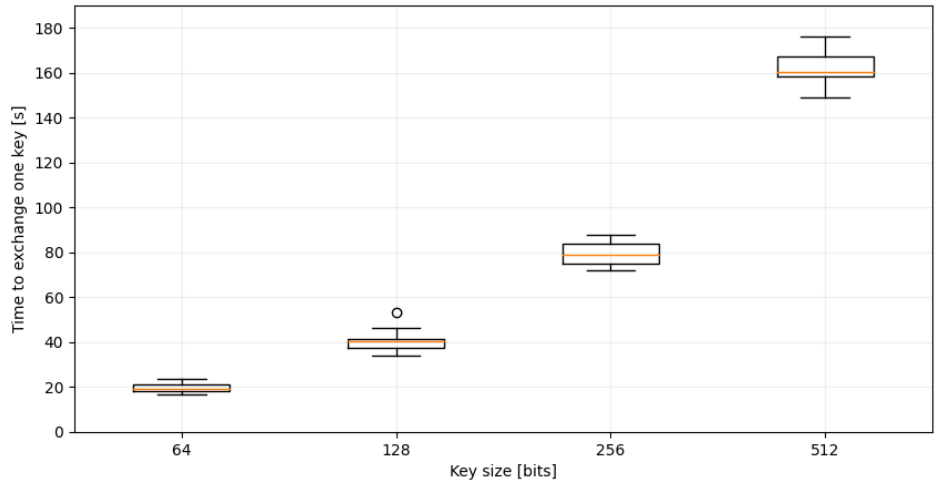
Figure 9: Sequence diagram of the steps followed in the functional validation.

The result of this validation was positive, proving the correct functioning of key formation processes and synchronization tasks.

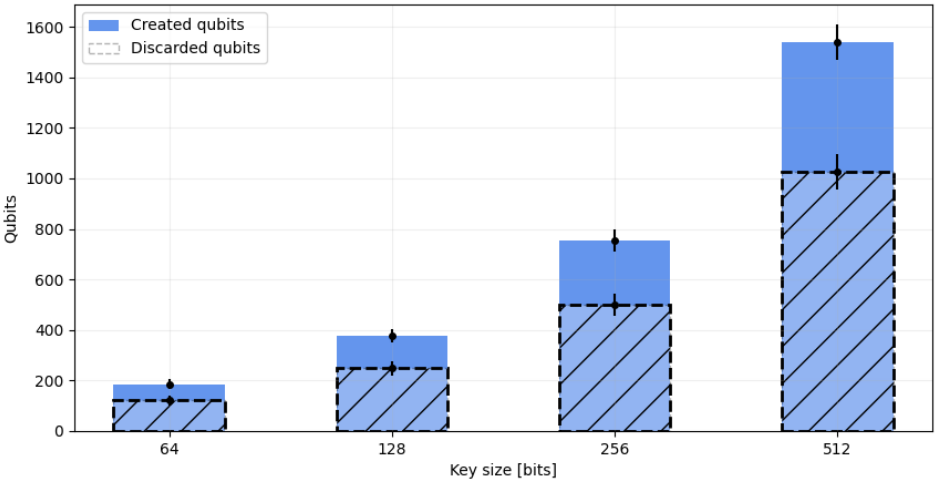
5.3 Performance Evaluation

To understand better the efficiency of our prototype, in terms of key exchange time and discarded qubits in the formation of a key, a performance evaluation has been conducted. This experiment was run over the same scenario as the previous experiment, shown in Figure 8.

In this evaluation, the application creates a key stream between neighboring modules, where the E91 protocol is implemented to form keys. It waits until 20 keys have been exchanged and retrieves them to get some information about the exchange times and number of discarded qubits per key. This test has been executed for different key lengths, in particular, 64, 128, 256, and 512-bit length keys have been used. The results of the evaluation are shown in Figure 10a,b.



(**a**)



(**b**)

Figure 10: Performance evaluation results. (**a**) Time spent in the exchange of one key as a function of the key size. For each key size, the distribution of the 20 exchanged keys is shown. (**b**) Qubits discarded when exchanging a key as a function of the key size. For each key size, the distribution of the 20 exchanged keys is shown.

The exchange time of a key increases linearly with the key size (note that the separations in the horizontal axis are not incremented linearly), increasing from as little as 20 s for a 64-bit key, to 2 min for a 512-bit key. However, these times do not necessarily match real-life times.

In the case of the number of discarded qubits, it also grows linearly for the key size (note again the separation in the horizontal axis), which was the expected result since, in our implementation of the E91 protocol, the average number of discarded qubits is $2/3$. In the implemented protocol, Alice and Bob use the same set of three bases. Consequently, there are 9 possible basis combinations, and only in $1/3$ of those combinations Alice and Bob choose the same measurement basis. Therefore, the number of qubits that

can be used to create the key is approximately $1/3$ of the total created qubits.That is exactly what can be seen in the bar plot of Figure 10b.

5.4 Performance of Orchestration Actions

In this experiment, the QKD network digital twin orchestrator builds digital twins of QKD networks with different numbers of nodes, making use of the open source network orchestration solution OSM, and collects the time spent in building the digital twins as a function of the number of nodes in the defined QKD network. A schematic of the scenario of this test can be seen in Figure 11.

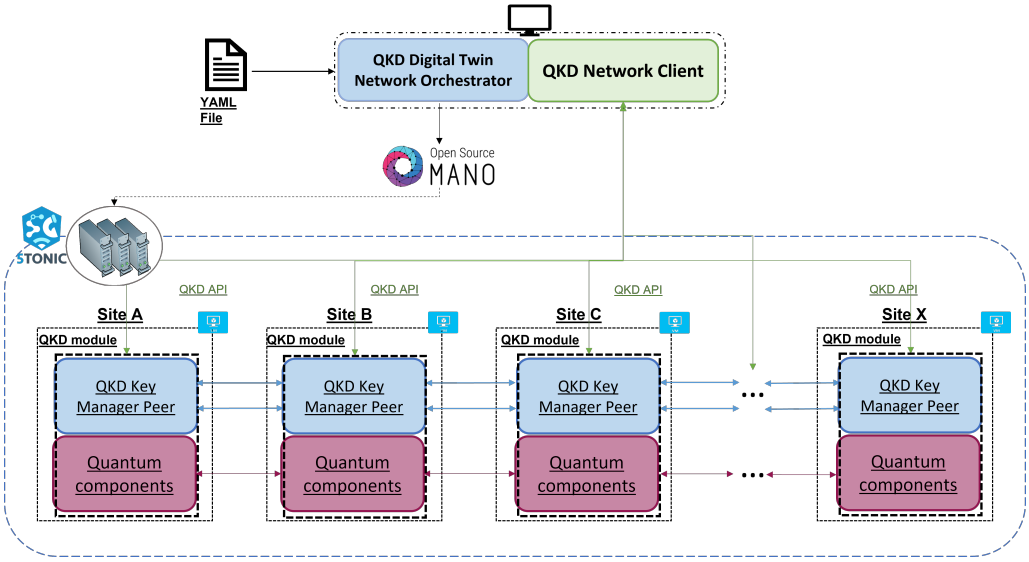


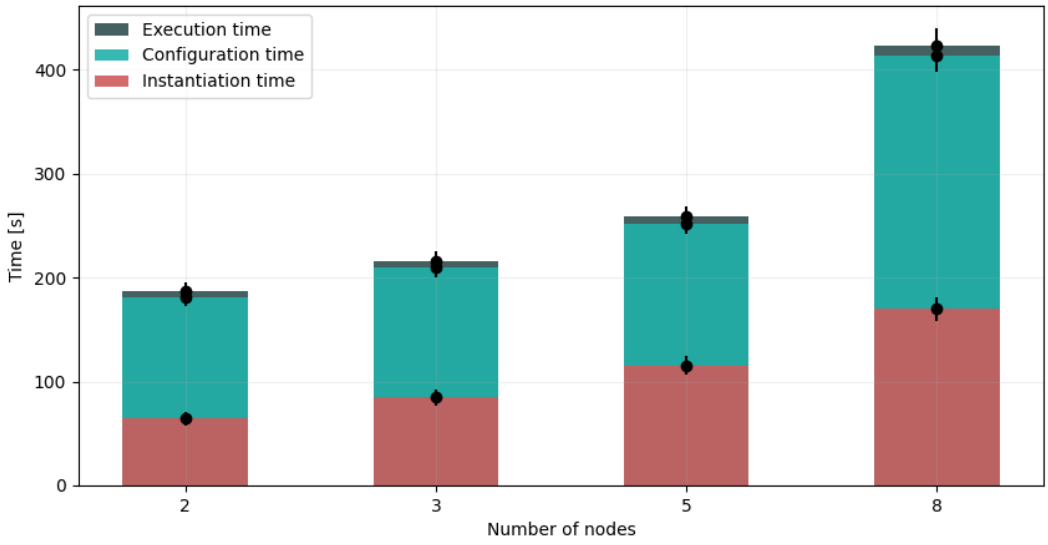
Figure 11: Scenario scheme of the experiment regarding the performance of orchestration actions.

To better analyze the results, the deployment time is divided into three main processes: the time of the virtual machine's instantiation, which corresponds to Step 2 defined in Section 3.1, the configuration time of those machines as QKD network sites, which includes the download of the necessary software, and the execution time, i.e., starting SimulaQron in each node to make the QKD network operational. These two processes conform the Step 3 in Section 3.1.

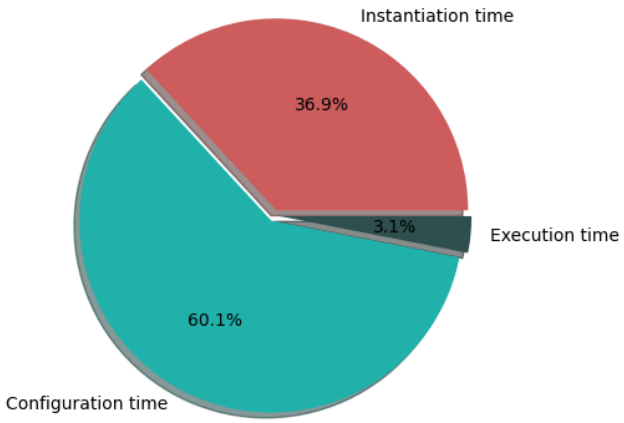
The first process conducted is the instantiation, which corresponds to the bottom range shown in Figure 12a. A linear growth can be seen in this component (note that the separation in the horizontal axis does not scale linearly) since the OSM executes the deployment sequentially.

The second process, the configuration, is represented in the middle range of Figure 12a, and as can be seen, it is the most time-consuming process. The behavior in the time spent performing this process can be explained because of the functioning of Ansible, discussed in Section 4.1. Since the default number of parallel processes that Ansible sends is 5 [49], the configuration time remains constant until this limit is exceeded, doubling in size once the number of nodes is greater than 5. As can be seen in Figure 12a, for networks of 1 to 5 nodes the configuration time is the same, while for the 8-node network under consideration, a growth of about twice as much is observed.

The execution time, which is the initialization of the emulator SimulaQron in each module, corresponds to the upper range in Figure 12a. It is quite small compared to the other two considered times and remains virtually constant for every considered number of nodes since it corresponds to the execution of a command to start SimulaQron, which is a fast process.



(**a**)



(**b**)

Figure 12: Deployment evaluation results. (**a**) Deployment time as a function of the network node number. For each network, the distribution of the 20 deployments is shown. (**b**) Average percentage that each process represents in a deployment.

Therefore, after analyzing the results, we can say that the deployment time consists mainly of two processes: one that grows linearly and corresponds to node instantiation, and another that involves node

configuration and increases for every group of five nodes due to the maximum number of parallelized tasks supported by Ansible [49].

As can be seen in Figure 12a, the service allows, for example, having an operational QKD network of 2 nodes in about 3 min, or a larger QKD network with eight nodes in less than 7 min. Thanks to this test, it has been proven that digital twins of QKD networks can be built and made operational in a reduced time-frame.

Figure 12b shows a diagram of the fractions each process represents of the total deployment time. As discussed, the configuration of the nodes represents more than half of the total time (60.1% of the total), followed by the instantiation process that takes 39.6% of the total time, and lastly, the execution, which corresponds roughly to the 3% of the total.

6 CONCLUSIONS AND FUTURE WORK

Quantum networks are a novel research topic expected to change and evolve significantly in the next years, bringing many scientific and security-related advantages. Awaiting the hardware to be good enough for feasible and accessible deployment of QKD networks, emulation services are a valuable option for testing and experimenting with QKD-related subjects. In this context, we have designed, implemented, and validated a solution that allows the automatic deployment of QKD networks digital twins [12]. The service virtualizes the sites of a QKD network in virtual machines, offering a cost-effective solution to the problem of physically building a QKD network. A QKD protocol is implemented using an open-source quantum emulator, SimulaQron, and a standardized API, the ETSI GS QKD 004 [27], is included to enable the communication between clients and networks, making the digital twin environment very similar to a real QKD network.

The tests that have been carried out show that the service is capable of building QKD network digital twins and making it operational in a reduced time frame that scales linearly with the number of nodes, achieving the deployment of an 8-node network in about 7 min (Figure 12a).

The current implementation of the service provides a solid software base on which to build applications that make use of the QKD network digital twin environment, this implementation will be extended with different features to help create more specific applications.

The emulated digital twin environment could be a useful tool not only for the development of QKD network technologies but also for the research on the integration of Quantum Communications in the current network since it can potentially be used on larger experiments leveraging on its orchestration features.

The presented service is, in summary, a novel solution that allows experimentation over a digital twin, a virtual replica, of a QKD network. This opens the door to many diverse research lines, from the study of the integration of QKD on current infrastructures to the analysis of different strategies for quantum key

exchange in multi-domain environments, i.e., environments where there may be networks from different providers. Some updates of the service are expected to be available soon, enabling features such as eavesdropping detection or hybrid deployments.

Our work in the short term will include the implementation of the complete E91 protocol, to ensure the realization of tests of the CHSH inequality, and thus enable eavesdropping verification in the communication processes. Other lines of research, such as the support of a backend that takes into account different quantum parameters, like decoherence or quality of entanglement, are already being developed. The deployment of hybrid environments where real and virtualized quantum equipment work together is also being explored.

Although the authentication of the channels is not addressed in this first version, the exploration of some options to solve that matter, such as Post-Quantum Cryptography (PQC) algorithms in the IPsec protocol [53] or methods rooted at physical properties [54] like Physical Unclonable Functions (PUFs) [55], is also a line of research that we intend to explore.

AUTHOR CONTRIBUTIONS

Software, R.M., B.N., and B.L.; Validation, R.M. and B.N.; Investigation, R.M., B.L., I.V., F.V. and B.N.; Writing – Original Draft Preparation, B.L., R.M., I.V., and B.N.; Writing – Review and Editing, B.L., I.V., F.V., and B.N.; Visualization, B.L.; Supervision, I.V. and F.V. All authors have read and agreed to the published version of the manuscript.

FUNDING

The work of Blanca Lopez has partially been supported by the MCIN with funding from European Union NextGenerationEU (PRTR-C17.I1) and funding from the Comunidad de Madrid - Programa de Acciones Complementarias, Madrid Quantum. This publication is also part of the project 6GINSPIRE PID2022-137329OB-C42, funded by MCIN/AEI/10.13039/501100011033/.

CONFLICTS OF INTEREST

The authors declare no conflicts of interest.

ABBREVIATIONS

The following abbreviations are used in this manuscript:

QKD	Quantum Key Distribution
NRBG	Non-deterministic Random Bit Generator
RSA	RivestShamirAdleman
DH	DiffieHellman
NFV	Network Functions Virtualization
5TONIC	5G Telefónica Open Network Innovation Centre
ETSI	European Telecommunications Standard Institute
IETF	Internet Engineering Task Force
API	Application Programming Interface
OSM	Open Source MANO
VIM	Virtual Infrastructure Manager
KSID	Key Stream ID
MANO	Management and Orchestration
PQC	Post-Quantum Cryptography
PUF	Physical Unclonable Function

REFERENCES

[1] Bayerstadler, A.; Becquin, G.; Binder, J.; Botter, T.; Ehm, H.; Ehmer, T.; Erdmann, M.; Gaus, N.; Harbach, P.; Hess, M.; et al. Industry quantum computing applications. *Epj Quantum Technol.* **2021**, *8*, 25.

[2] Emani, P.S.; Warrell, J.; Anticevic, A.; Bekiranov, S.; Gandal, M.; McConnell, M.J.; Sapiro, G.; Aspuru-Guzik, A.; Baker, J.T.; Bastiani, M.; et al. Quantum computing at the frontiers of biological sciences. *Nat. Methods* **2021**, *18*, 701–709.

[3] Rivest, R.L.; Shamir, A.; Adleman, L. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM* **1978**, *21*, 120126. https://doi.org/10.1145/359340.359342.

[4] Diffie, W.; Hellman, M. New directions in cryptography. *IEEE Trans. Inf. Theory* **1976**, *22*, 644–654. https://doi.org/10.1109/TIT.1976.1055638.

[5] Shor, P. Algorithms for quantum computation: Discrete logarithms and factoring. In Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Washington, DC, USA, 20–22 November 1994; pp. 124–134. https://doi.org/10.1109/SFCS.1994.365700.

[6] Chen, Y.A.; Zhang, Q.; Chen, T.Y.; Cai, W.Q.; Liao, S.K.; Zhang, J.; Chen, K.; Yin, J.; Ren, J.G.; Chen, Z.; et al. An integrated space-to-ground quantum communication network over 4600 kilometres. *Nature* **2021**, *589*, 214219. https://doi.org/10.1038/s41586-020-03093-8.

[7] Wengerowsky, S.; Joshi, S.K.; Steinlechner, F.; Zichi, J.R.; Dobrovolskiy, S.M.; van der Molen, R.; Los, J.W.N.; Zwiller, V.; Versteegh, M.A.M.; Mura, A.; et al. Entanglement distribution over a 96-km-long submarine optical fiber. *Proc. Natl. Acad. Sci. USA* **2019**, *116*, 6684–6688. https://doi.org/10.

1073/pnas.1818752116.

[8] Lopez, D.; Brito, J.P.; Pastor, A.; Martin, V.; Sánchez, C.; Rincon, D.; Lopez, V. Madrid Quantum Communication Infrastructure: A testbed for assessing QKD technologies into real production networks. In Proceedings of the 2021 Optical Fiber Communications Conference and Exhibition (OFC), Washington, DC, USA, 6–11 June 2021; pp. 1–4.

[9] Zhou, L.; Lin, J.; Xie, Y.M.; Lu, Y.S.; Jing, Y.; Yin, H.L.; Yuan, Z. Experimental Quantum Communication Overcomes the Rate-Loss Limit without Global Phase Tracking. *Phys. Rev. Lett.* **2023**, *130*, 250801. https://doi.org/10.1103/PhysRevLett.130.250801.

[10] Grünenfelder, F.; Boaron, A.; Resta, G.V.; Perrenoud, M.; Rusca, D.; Barreiro, C.; Houlmann, R.; Sax, R.; Stasi, L.; El-Khoury, S.; et al. Fast single-photon detectors and real-time key distillation enable high secret-key-rate quantum key distribution systems. *Nat. Photonics* **2023**, *17*, 422–426.

[11] Li, W.; Zhang, L.; Tan, H.; Lu, Y.; Liao, S.K.; Huang, J.; Li, H.; Wang, Z.; Mao, H.K.; Yan, B.; et al. High-rate quantum key distribution exceeding 110 Mb s–1. *Nat. Photonics* **2023**, *17*, 416–421.

[12] Networks it uc3m. QKD Digital Twin Service (QDTS). Available online: https://github.com/Networks-it-uc3m/QDTS (Accessed on 24 January 2024).

[13] Tuegel, E.J.; Ingraffea, A.R.; Eason, T.G.; Spottswood, S.M. Reengineering aircraft structural life prediction using a digital twin. *Int. J. Aerosp. Eng.* **2011**, *2011*, 154798.

[14] Almasan, P.; Ferriol-Galmés, M.; Paillisse, J.; Suárez-Varela, J.; Perino, D.; López, D.; Perales, A.A.P.; Harvey, P.; Ciavaglia, L.; Wong, L.; et al. Digital twin network: Opportunities and challenges. *arXiv Prepr.* **2022**, arXiv:2201.01144.

[15] Singh, M.; Fuenmayor, E.; Hinchy, E.P.; Qiao, Y.; Murray, N.; Devine, D. Digital Twin: Origin to Future. *Appl. Syst. Innov.* **2021**, *4*. https://doi.org/10.3390/asi4020036.

[16] Vaezi, M.; Noroozi, K.; Todd, T.D.; Zhao, D.; Karakostas, G.; Wu, H.; Shen, X. Digital Twins From a Networking Perspective. *IEEE Internet Things J.* **2022**, *9*, 23525–23544. https://doi.org/10.1109/JIOT.2022.3200327.

[17] Barricelli, B.R.; Casiraghi, E.; Fogli, D. A Survey on Digital Twin: Definitions, Characteristics, Applications, and Design Implications. *IEEE Access* **2019**, *7*, 167653–167671. https://doi.org/10.1109/ACCESS.2019.2953499.

[18] Wootters, W.K.; Zurek, W.H. A single quantum cannot be cloned. *Nature* **1982**, *299*, 802803. https://doi.org/10.1038/299802a0.

[19] Horodecki, R.; Horodecki, P.; Horodecki, M.; Horodecki, K. Quantum entanglement. *Rev. Mod. Phys.* **2009**, *81*, 865.

[20] Clauser, J.F.; Horne, M.A.; Shimony, A.; Holt, R.A. Proposed Experiment to Test Local Hidden-Variable Theories. *Phys. Rev. Lett.* **1969**, *23*, 880–884. https://doi.org/10.1103/PhysRevLett.23.880.

[21] Nurhadi, A.I.; Syambas, N.R. Quantum Key Distribution (QKD) Protocols: A Survey. In Proceedings of

the 2018 4th International Conference on Wireless and Telematics (ICWT), Nusa Dua, Bali, Indonesia, 12–13 July 2018; pp. 1–5. https://doi.org/10.1109/ICWT.2018.8527822.

[22] Bennett, C.H.; Brassard, G. Quantum cryptography: Public key distribution and coin tossing. *arXiv Prepr.* **1984**, arXiv:2003.06557.

[23] Ekert, A.K. Quantum cryptography based on Bell's theorem. *Phys. Rev. Lett.* **1991**, *67*, 661–663. https://doi.org/10.1103/PhysRevLett.67.661.

[24] ETSI Industry Specification Group (ISG) on Quantum Key Distribution (QKD). Available online: https://www.etsi.org/committee/1430-qkd(Accessedon23October2023).

[25] IETF Quantum Internet Research Group (QIRG). Available online: https://datatracker.ietf.org/group/qirg/about/ (Accessed on 27 October 2023).

[26] Kozlowski, W.; Wehner, S.; Meter, R.V.; Rijsman, B.; Cacciapuoti, A.S.; Caleffi, M.; Nagayama, S. *Architectural Principles for a Quantum Internet*; RFC 9340; RFC Editor: 2023.

[27] ETSI. *Quantum Key Distribution (QKD); Application Interface*; ETSI GS QKD 004 V2.1.1 (2020-08); ETSI: Sophia Antipolis, France, 2020.

[28] ETSI. *Quantum Key Distribution (QKD); Protocol and Data Format of REST-Based Key Delivery API*; ETSI: Sophia Antipolis, France, 2019.

[29] Wang, C.; Rahman, A.; Li, R.; Aelmans, M.; Chakraborty, K. Application Scenarios for the Quantum Internet; Internet Engineering Task Force. 2023. Work in Progress. Available online: https://datatracker.ietf.org/doc/draft-irtf-qirg-quantum-internet-use-cases/ (Accessed on 24 January 2024).

[30] ETSI. *Quantum Key Distribution (QKD); Device and Communication Channel Parameters for QKD Deployment*; ETSI GS QKD 012 V1.1.1 (2019-02); ETSI: Sophia Antipolis, France, 2020.

[31] Lopez, V.; Pastor, A.; Lopez, D.; Aguado, A.; Martin, V. Applying QKD to improve next-generation network infrastructures. In Proceedings of the 2019 European Conference on Networks and Communications (EuCNC), Valencia, Spain, 18–21 June 2019; pp. 283–288. https://doi.org/10.1109/EuCNC.2019.8802060.

[32] Mehic, M.; Niemiec, M.; Rass, S.; Ma, J.; Peev, M.; Aguado, A.; Martin, V.; Schauer, S.; Poppe, A.; Pacher, C.; et al. Quantum Key Distribution: A Networking Perspective. *ACM Comput. Surv.* **2020**, *53*, 1–41. https://doi.org/10.1145/3402192.

[33] Cao, Y.; Zhao, Y.; Wang, Q.; Zhang, J.; Ng, S.X.; Hanzo, L. The Evolution of Quantum Key Distribution Networks: On the Road to the Qinternet. *IEEE Commun. Surv. Tutorials* **2022**, *24*, 839–894. https://doi.org/10.1109/COMST.2022.3144219.

[34] Aji, A.; Jain, K.; Krishnan, P. A Survey of Quantum Key Distribution (QKD) Network Simulation Platforms. In Proceedings of the 2021 2nd Global Conference for Advancement in Technology (GCAT), Bangalore, India, 1–2 October 2021, pp. 1–8. https://doi.org/10.1109/GCAT52182.2021.9587708.

[35] Wu, X.; Chung M., J.F.; Kolar, A.; Wang, E.; Zhong, T.; Kettimuthu, R.; Suchara, M. Simulations of Photonic Quantum Networks for Performance Analysis and Experiment Design. In Proceedings of the 2019 IEEE/ACM Workshop on Photonics-Optics Technology Oriented Networking, Information and Computing Systems (PHOTONICS), Denver, CO, USA, 18 November 2019; pp. 28–35. https://doi.org/10.1109/PHOTONICS49561.2019.00010.

[36] Diadamo, S.; Nötzel, J.; Zanger, B.; Bee, M.M. QuNetSim: A Software Framework for Quantum Networks. *IEEE Trans. Quantum Eng.* **2021**, *2*, 1–12. https://doi.org/10.1109/TQE.2021.3092395.

[37] Coopmans, T.; Knegjens, R.; Dahlberg, A.; Maier, D.; Nijsten, L.; de Oliveira Filho, J.; Papendrecht, M.; Rabbie, J.; Rozpedek, F.; Skrzypczyk, M.; et al. NetSquid, a discrete-event simulation platform for quantum networks. *Commun. Phys.* **2021**, *4*, 164. [arXiv:quant-ph/2010.12535]. https://doi.org/10.1038/s42005-021-00647-8.

[38] Wu, X.; Zhang, B.; Jin, D. Parallel Simulation of Quantum Key Distribution Networks. In Proceedings od the 2020 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, Miami, FL, USA, 15–17 June 2020; pp. 187–196. https://doi.org/10.1145/3384441.3395988.

[39] Dahlberg, A.; Wehner, S. SimulaQron—A simulator for developing quantum internet software. *Quantum Sci. Technol.* **2018**, *4*, 015001. https://doi.org/10.1088/2058-9565/aad56e.

[40] Condoluci, M.; Mahmoodi, T. Softwarization and virtualization in 5G mobile networks: Benefits, trends and challenges. *Comput. Netw.* **2018**, *146*, 65–84. https://doi.org/https://doi.org/10.1016/j.comnet.2018.09.005.

[41] Morabito, R.; Kjällman, J.; Komu, M. Hypervisors vs. Lightweight Virtualization: A Performance Comparison. In Proceedings of the 2015 IEEE International Conference on Cloud Engineering, Tempe, AZ, USA, 9–13 March 2015; pp. 386–393. https://doi.org/10.1109/IC2E.2015.74.

[42] Mijumbi, R.; Serrat, J.; Gorricho, J.; Bouten, N.; De Turck, F.; Boutaba, R. Network Function Virtualization: State-of-the-Art and Research Challenges. *IEEE Commun. Surv. Tutorials* **2016**, *18*, 236–262. https://doi.org/10.1109/COMST.2015.2477041.

[43] ETSI Open Source MANO. Available online: https://osm.etsi.org/ (Accessed on 11 October 2023).

[44] Cloudify. A Platform that Turns Clouds, Tools & Technologies into Self-Managed Environments. 2023. Available online: https://cloudify.co (Accessed on 15 November 2023).

[45] OpenStack. Build the Future of Open Infrastructure. 2023. Available online: https://www.openstack.org/ (Accessed on 15 November 2023).

[46] The Linux Foundation. Kubernetes: Production-Grade Container Orchestration. 2023. Available online: https://kubernetes.io (Accessed on 15 November 2023).

[47] Liu, G.; Huang, B.; Liang, Z.; Qin, M.; Zhou, H.; Li, Z. Microservices: Architecture, container, and challenges. In Proceedings of the 2020 IEEE 20th International Conference on Software Quality, Re-