Quantum Key Distribution with Trusted Relay using an ETSI-compliant Software-Defined Controller

Riccardo Bassi, Qiaolun Zhang, Alberto Gatto, Massimo Tornatore, Giacomo Verticale
Politecnico di Milano
name.surname@polimi.it

***Abstract*—Quantum Key Distribution (QKD) is a mechanism that allows two entities to agree on a secret key over a quantum channel. Recently, commercially viable QKD technology appeared on the market and the European Telecommunications Standards Institute (ETSI) established a working group to standardize QKD for commercial applications. Several testbeds and pre-commercial QKD networks are being deployed in the world, all requiring sophisticated control mechanisms to enable key exchanges between non-adjacent nodes for remote applications. In this paper, we implement and evaluate a prototype that realizes the architecture defined by ETSI and uses trusted relay nodes to exchange keys between non-adjacent nodes. In particular, we present the Software-Defined Network (SDN) Controller and the Software-Defined QKD nodes designed to control the QKD experimental testbed in Milan, named PoliQI. We test the proposed prototype in an emulation environment, showing its advantages in inter-datacenter communication scenarios in terms of key delivery time and application acceptance ratio.**

I. Introduction

Quantum-communication technologies are becoming commercially viable and are spurring the interest of many companies and governments worldwide. More specifically, Quantum Key Distribution (QKD) is a family of protocols designed to use physical-layer technologies to securely exchange keys that can be used to protect communications from eavesdropping. Since QKD is resistant to post-quantum attacks (i.e., to attacks using a quantum computer), it can provide quantum-resistant security to legacy applications relying on non-quantum-resistant algorithms.

QKD has already been tested and verified in optical networks to exchange keys between adjacent nodes [1]. To enable connections between non-adjacent nodes, QKD network is currently being investigated and several testbeds have been proposed [2]–[4]. Moreover, QKD is being standardized by the European Telecommunications Standards Institute (ETSI). Specifically, the ETSI QKD committee is currently releasing a set of technical specifications outlining a control plane for QKD network based on the principles of Software-Defined Networking [5]. However, QKD-network testbeds require sophisticated control mechanisms to exchange keys for remote applications, and control technology to seamlessly facilitate communications within QKD networks using ETSI standards is at an initial stage.

To address these issues, this work aims to design an ETSI-compliant QKD network control architecture on top of the PoliQI network, a QKD testbed currently under development in the metropolitan area of Milan, Italy. PoliQI will provide high-speed, quantum-secure inter-datacenter communication among five datacenters operated by government, military agencies, telecommunication operators, financial companies and universities [6]. The PoliQI network will support QKD between adjacent nodes and will also provide two mechanisms for key exchange between non-adjacent nodes, namely *trusted relays* and *optical bypass* [7]. In this work, we focus on trusted relays as they allow to achieve higher key rates compared to optical bypass and they will be deployed earlier than optical bypass due to their lower implementation complexity. The challenge of the PoliQI project is to seamlessly integrate the QKD nodes and procedures of optical bypass and trusted relay into the operations of the datacenters, which must adhere to strict safety and security standards and must operate at high speed and low latency with high workloads. One key element for seamless integration of such a disruptive technology into datacenters is the adoption of standard interfaces.

The proposed QKD network prototype comprises QKD links, Software Defined QKD (SD-QKD) Nodes, and an SDN Controller. Each QKD link contains multiple quantum channels (QCs), where qubits are transmitted. The SD-QKD Node is internally composed of a Key Management Entity (KME) and an SDN Agent. Each local node maintains a local key database, namely, quantum key pool (QKP). The network can also pre-distribute keys in QKP to reduce latency at the cost of some network capacity. The KME is the one in charge of handling the key material that comes from the QCs. The SDN Agent is responsible for the interactions with the SDN Controller. The SDN controller is designed to perform two main tasks: i) to monitor the status of the QCs (e.g., to verify the key generation rate and the amount of keys stored), and ii) to collect new application requests and decide whether to serve the new application. The prototype implements the ETSI standard interfaces between the SDN Controller and the SD-QKD Nodes and between the SD-QKD Nodes and the Applications.

The contributions of this paper can be summarized as follows.

This project was partially funded by the Italian project Quancom. This work was also partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART"). Moreover, this work was also partly supported by the Italian Ministry of University and Research (MUR) and the European Union (EU) under the PON/REACT project.

978-1-6654-7598-3/23/$31.00 © 2023 IEEE

- We propose a prototype implementation of an ETSI-compliant SDN architecture for controlling a QKD network, which seamlessly integrates QKD into confidential communications.
- We extend the ETSI architecture to support QKD between non-adjacent nodes using Trusted Nodes.
- We develop an efficient algorithm for deciding whether to admit a new application without hampering the ability of other applications to periodically renew their keys.
- We validate the performance of the proposed prototype with extensive simulations and identify some criticalities and directions for improvement.

The paper is organized as follows. Section II reviews some related work on QKD network testbeds and their control mechanisms. Section III overviews the network architecture. Section IV discusses the proposed key management scheme and key relay process. Finally, we analyse the performance of the proposed prototype in Section V and conclude the paper in Section VI.

II. Related Work

Research on quantum communication technology has gained traction in the last few years. Various QKD network architectures and testbeds are being implemented. Within the existing QKD network architectures, two practical techniques have been identified to distribute keys between non-adjacent nodes: optical bypass and trusted relay. [7] The optical bypass technique enables key distribution between intermediate nodes by setting up a QC through any intermediate nodes. The trusted relay technique uses uncompromised intermediate nodes to relay the keys for non-adjacent nodes. The latter technique can achieve a higher key rate compared to optical bypass since the signal does not suffer from attenuation along the whole path. Other techniques, such as untrusted relays and quantum repeater [8] are not discussed in our work as they are not mature enough for practical deployments.

QKD has been already investigated within various QKD network testbed [2]–[4]. The authors of [4] demonstrate secure connections between eight users in a metropolitan network without trusted relays. The authors of [9] propose an architecture for a Software-Defined QKD network and the authors of [3], [4] implement key management schemes to utilize trusted relays to scale the QKD distance. However, these works do not consider building ETSI-compliant QKD controllers and do not consider key management schemes to accommodate a heavy load of key requests. In a preliminary work [10], we implemented a prototype that allows two secure application entities (SAEs) to establish a secure communication channel through the keys delivered by the KMEs. However, this implementation works only in the simplest use case in which there are only two KMEs directly connected by a quantum channel.

Different from previous works, our main goal is to develop a control architecture that allows to seamlessly facilitate communications within an ETSI-compliant QKD network. In addition, we extend the ETSI architecture for exchanging keys between non-adjacent nodes, and we developed an efficient algorithm for key request admission without jeopardizing the rekeying process of existing services.

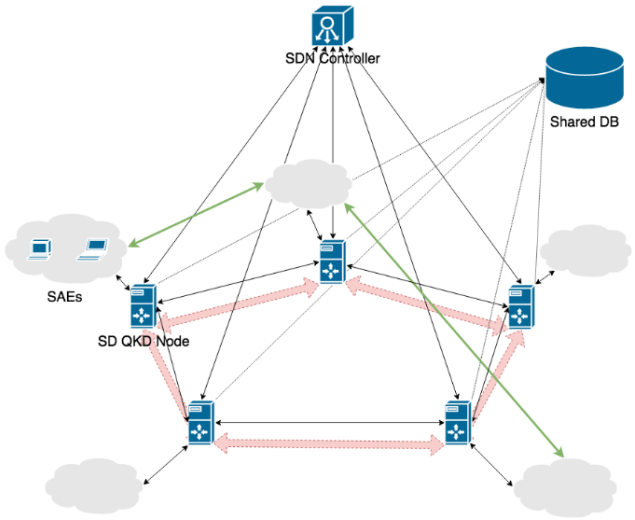

Fig. 1. The PoliQI infrastructure showing the SDN control entities. The red arrows represent the QCs; the green ones represent the secured communication channels established among SAEs; the solid black ones represent which components can communicate through RESTApi interfaces; the dashed black ones indicate the nodes that interact with the shared database.

III. Network Architecture Overview

This section first introduces the different network elements of the proposed QKD network, and then discusses its main design principles.

The PoliQI testbed, depicted in Fig. 1 includes the following elements.

a) SAEs: they are the application components that request and use the secure keys. The master SAE initiates the request, while the slave SAE operates on the input of the master SAE. Periodically, the SAEs perform rekeying according to the application security needs.

b) SDN Controller: it controls all the SD-QKD nodes in the network and has perfect knowledge of their states including all the key requests coming from the SAEs and the key generation rate of the QCs.

c) QC: it is the channel over which the QKD protocol is run. We assume that there is also a classical channel associated with it over which the classical part of the QKD protocol is run. Each QC automatically generates key material and securely delivers it to the two KMEs at the sides of the QC.

d) SD-QKD Node: it manages the key material coming from the QC and uses it to generate and distribute the keys to the SAEs. It is internally divided into the four main sub-components shown in Fig. 2: the KME, the SDN Agent, the local database, and the quantum module. The **KME** gathers the key material from the QC, generates the application keys and delivers them to the SAEs. The **SDN Agent** is responsible for communications with the SDN Controller. The **local database** stores the key material together with information about the

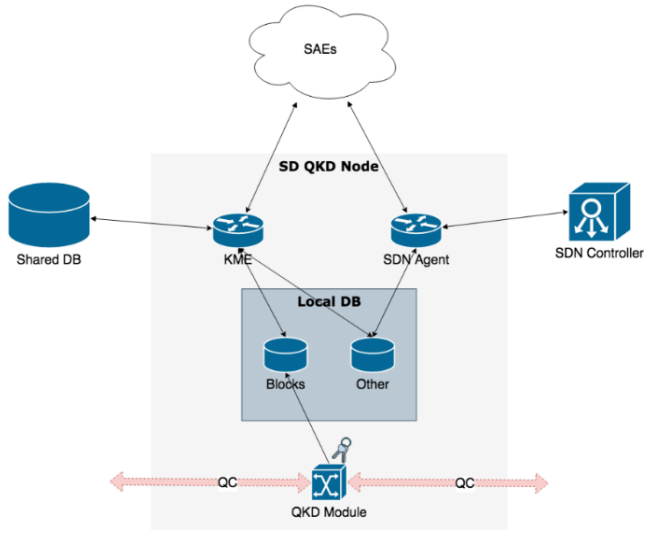

Fig. 2. The SD-QKD Node internal structure and its connections.

SAEs; it also stores any cached keys for usage in the *QKP* mode described in Section IV. The **quantum module** executes the QKD protocol with the adjacent nodes.

e) Shared database: it stores the instructions needed by the KME to generate the correct application keys from the key material exchanged over the QC.

The elements in the network architecture introduced above are implemented as python modules as shown in Fig. 3. The grey parts, namely, *Load Generator Module* and *Quantum Channel Simulation Module*, are used in the testing and will be substituted with the actual applications and quantum channel in the PoliQI testbed. In this work, we implemented *SAE Module*, *SDN Controller Module*, and *QKD Node Module*. The *SDN Controller Module* interacts with *SAE Module* to get key rate requests and assign keys to applications. In the meantime, the *SDN Controller Module* also interacts with the *QKD Node Module* to control the key generation and key distribution with trusted relay.

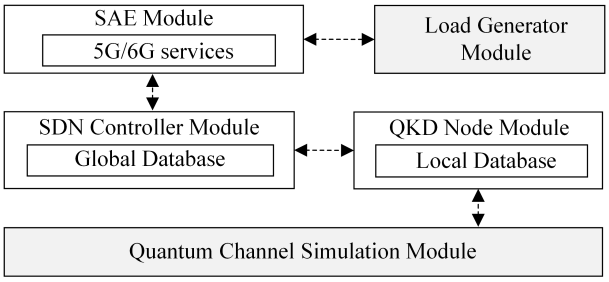

Fig. 3. Implemented modules and their relationships.

IV. Key Management Scheme

When a new application starts, it sets up a key session and requests an initial key. If the controller grants this new session, then the application will use the first key for securing its communications. Depending on the security requirements, this key can be used to encrypt an amount of data equal to the key size or it can be expanded and used to encrypt a larger amount of data, for example using it as a session key in the Transport Layer Security (TLS) protocol. In the latter case, the application relinquishes the perfect security model but transmits at a higher rate. In either case, the application will periodically ask for a new key by means of a rekeying process. A failure in granting the initial key or a subsequent key has a different impact on the application. In the former case, the application cannot proceed and is forced to retry at a later time. In the latter case, the application could continue using the previous key, if this is allowed by the security rules, or it must stop. In this paper, we assume that the security rules make applications stop in case of a rekeying failure and that this event is less desirable to the end users than failure to grant an initial key.

A. Key Session Initialization

One of the most crucial steps in key management is the application registration by which a pair of SAEs set up a new identifier and reserve some resources (Fig. 4). The key request initialization process is shown in Fig. 4. Let's assume that we want to exchange keys between a master SAE and a slave SAE. Node A and node B are the local QKD nodes connected to the master SAE and slave SAE, respectively. Each SAE is characterized by a unique SAE ID, whose assignment of SAE IDs is outside of the ETSI Standard [11]. In PoliQI each SAE randomly generates a Universally Unique IDentifiers according to RFC 4122 [12].

The master SAE initiates secure communications by sending an `ask_connection` message to the slave SAE, which includes the SAE ID of the master SAE. The slave SAE, in turn, opens a session with the local QKD node (Node B in the figure) using the `open_key_session` message. In turn, the local QKD node B registers the session with the SDN controller using the `new_app` message.

Upon success, the slave SAE answers the initial `ask_connection` message from the master SAE providing its own SAE ID. Then, the master SAE repeats the procedure with its own local QKD node (Node A in Figure 4). When both SAEs complete their registration, the Controller assigns the Key Stream ID (KSID) to the applications.

We define the time of key request initialization as *key request initialization time*, which contains the time that each component (QC Nodes and Controller) requires to accept (or reject) a new session. Specifically, the time of opening a session between QKD node B and slave SAE and registering a session between QKD node B and Controller is denoted with t_1 and t_2, respectively, as shown in Fig. 4. In addition, the time of opening a session between QKD node A and master SAE and registering a session between QKD node A and Controller is denoted with t_3 and t_4, respectively.

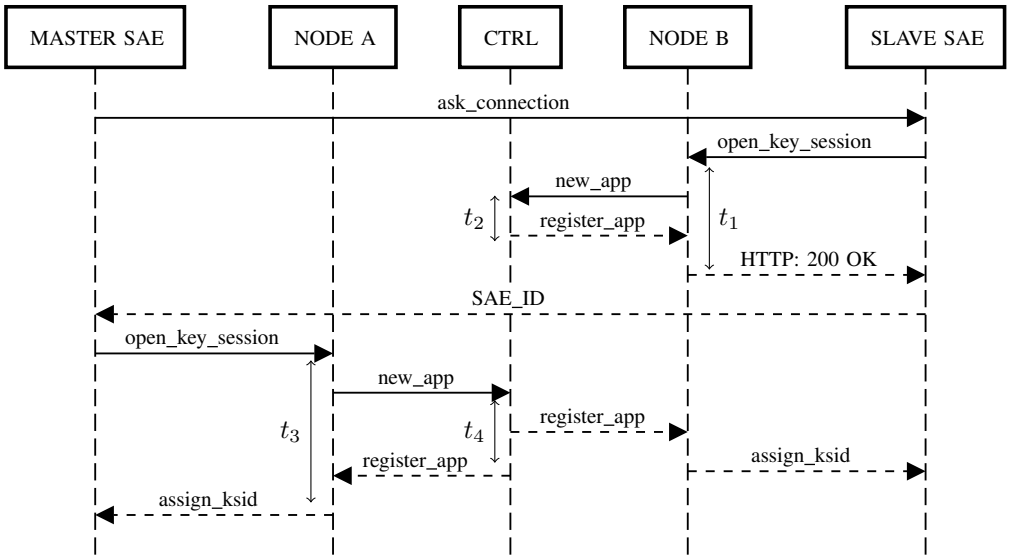

Fig. 4. Key request initialization, annotated with the times t_1, t_2, t_3, and t_4 measured in Section V-B

B. Session Admission Control

The SDN controller checks if the QKD nodes are directly connected; if not, the controller chooses a path of trusted QKD nodes and computes the amount of quantum key material needed to relay the keys along the path. Then, it computes the average amount of key material that is generated in the Quantum Channels along the path and verifies that the new request can be accommodated without jeopardizing existing sessions.

Let ℓ be the size of the key requested by the new application. We assume that time is divided into discrete time slots and key requests are processed at the end of each slot. Each node measures the amount of key material generated on each QC in each slot and reports it to the controller. Let $\overline{\Lambda}_j^{qc}(n)$ be the amount of key material generated on the j-th QC in the n-th interval. If the key material generated in an interval is not used to generate application keys, it is stored and used in the following slots or discarded after TTL slots. Let $\Lambda_i^a(n)$ be the size of the key requested by application i in the n-th slot and let $\delta(i,j) = 1$ if the application i uses material from the QC j or zero otherwise. Let $\mathcal{QC}'$ be the set of QC used by the new application and let $\mathcal{I}$ be the set of all the active applications.

In our prototype, the SDN controller accepts a session if the following equation is satisfied:

$$\min_{j\in\mathcal{QC}'} \sum_{n=n_0-\mathsf{TTL}}^{n_0} \left(\overline{\Lambda}_j^{qc}(n) - \sum_{i\in\mathcal{I}} \delta(i,j)\Lambda_i^a(n) \right) \geq \ell \qquad (1)$$

where n_0 is the current slot.

The SDN controller locks the topology database during these checks and, thus, can process one new session at a time.

C. Single-Hop Key Retrieval

When the Key Request Initialization is successfully completed, the Master SAE can request the first key starting the protocol in Figure 5 The Master SAE sends the local node a `get-key` message providing the pair of SAE identifiers and the requested key size. The local node (Node A in the Figure) allocates the requested number of bits from the local pool and sends to the global database the necessary information to repeat the same allocation at the remote node. Such information consists of the starting and ending position of the application key in the key stream generated from the QC. Upon success, the Master SAE receives the key and a key ID, which it sends to the remote SAE using an `ask-key` message. The Slave SAE makes a request to the local node, which retrieves the key information from the global database, extracts the key from its key pool and gives it to the Slave SAE. Periodically, the Master SAE performs the rekeying procedure, which consists in requesting a new key with the same protocol.

If the local key pool does not have enough bits to extract the requested key, the protocol fails and we register a *key error*. It is up to the SAEs to decide whether to continue using the old key or to interrupt the communications.

D. Key Relay Process

Figure 6 shows the key relay process in the multi-hop case. In the figure, the two SAEs are connected to node A and node D, respectively. The path passes through node B and node C.

The first step is the generation of a local key on each Quantum Channel with the protocol described in the previous section. These are represented by the blue, green, and red keys in Figure 6. Then, the key generated in the first link is relayed along the following links in the path up to the final node. At each hop, the key is encrypted with the local key using a simple One-Time Pad (OTP).

Once the key is successfully shared between the first and the last node, the first node generates a new key using the QKD process on the first link, encrypts it with the shared

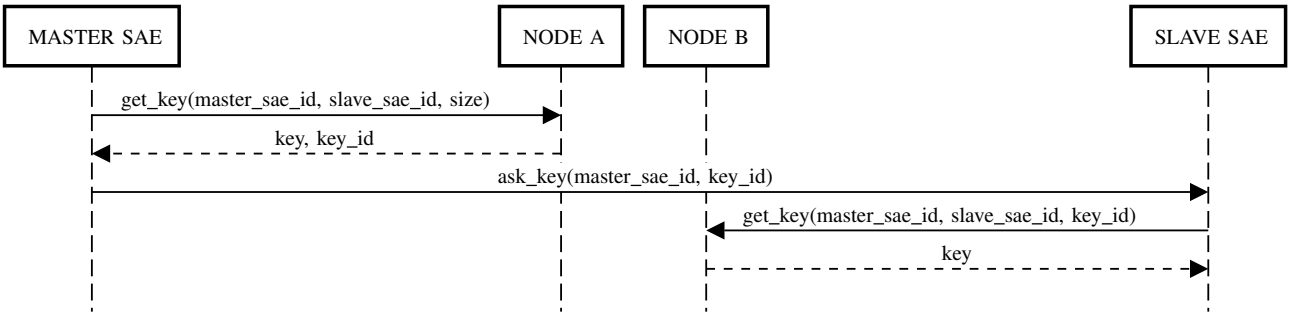

Fig. 5. The Key Retrieval Protocol

key using OTP and sends it to the last node over a classic channel. This procedure is called *basic* mode and results in a single key shared between the first and last node at the end of the exchange. Since this process is expensive and time-consuming, we also consider an alternative mode called *quantum key pooling (QKP)* in which the first node generates multiple keys in the first link and sends them to the last node in a single message. The keys that are not immediately necessary are locally stored in the nodes in case an SAE requests them, for example when rekeying. If these keys are not used, after TTL slots they are dropped. It is worth noting that in our prototype we use a single key shared between the first and last node to encrypt four new keys. This means that it is not possible to use OTP to relay keys on a multi-hop path, but it is necessary to use some semantically secure encryption scheme such as AES-CCM. Consequently, QKP does not guarantee perfect secrecy.

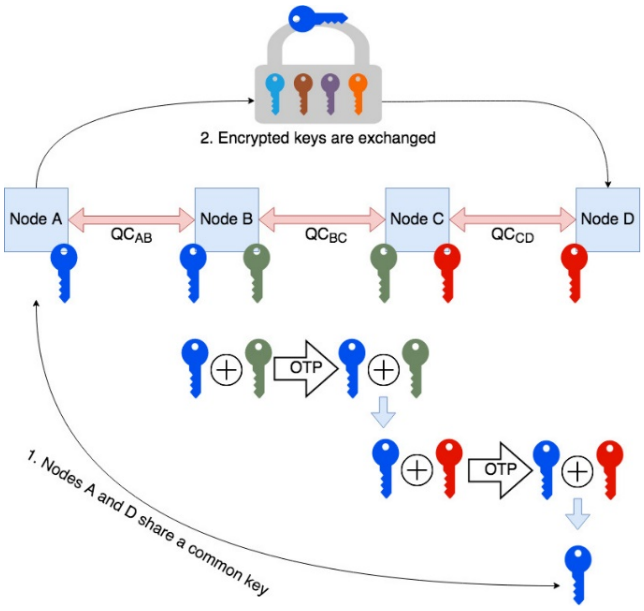

Fig. 6. Relaying process to have two non-adjacent nodes to share keys. In *BASIC* mode just one key is relayed, while in *QKP* mode the keys are multiple.

V. Illustrative Numerical Results

A. Evaluation Setting

We tested the proposed prototype in an emulation environment to assess its suitability for PoliQi, which focuses on inter-datacenter communication.The various components are run as concurrent threads on a Macbook with Intel i5 CPU (2.6 GHz), 8 GB of memory. The considered QKD network topology is the PoliQi topology as shown in Figure 1. QCs generate a random number of bytes uniformly distributed between 33 byte/s and 47 byte/s, which is equivalent to, on average, 320 bit/s. Every application requests a key of 128 bits every 5 seconds. Requests are randomly selected among all the node pairs and the SAEs are randomly distributed on the nodes, making sure that every node has at least one SAE. The time-to-live (TTL) is set to 15 s.

B. Key Request Initialization Time Breakdown

We first measure the required time of different components for key request initialization, namely the interval t_1 to t_4 described in Section IV-A. These simulations consist in performing 50 key request initializations with a very low load in order to have negligible congestion. The average measurements are reported in Table I.

The key request initialization time can be split into two parts, namely the *Slave SAE* part and *Master SAE* part. The *Slave SAE* part takes only 60 ms, since it only informs the Controller of a new application, without allocating any resources. For the *Master SAE* part, on the contrary, the SDN Controller becomes the bottleneck and takes 170 ms, of which most of the time is consumed by the interaction between node A and SDN Controller (140 ms), because it needs to check the resource availability along the path and inform all the nodes before the *KSID* is sent to the SAEs.

TABLE I
Breakdown of Key Request Initialization Times

Key Request	QC Node	Controller
Slave SAE	$t_1 = 60\,\text{ms}$	$t_2 = 30\,\text{ms}$
Master SAE	$t_3 = 170\,\text{ms}$	$t_4 = 140\,\text{ms}$

C. Key Delivery Time

Now we evaluate the key delivery time, which is defined as the time between the arrival of a new key request (performed by a master SAE) at a node and the time it returns the key. We tested *BASIC* mode and *QKP* mode without application rejection since no application is rejected at such low workloads.

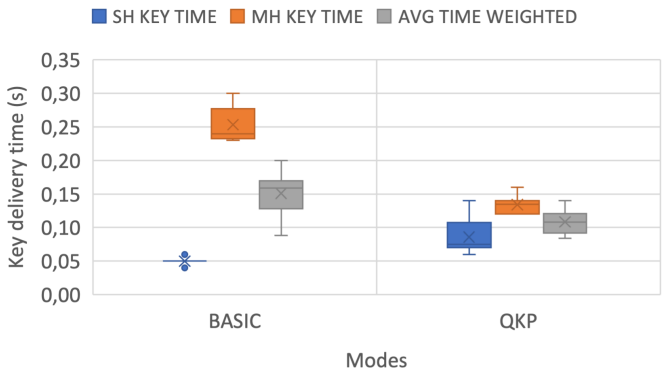

Fig. 7. Key delivery time.

Figure 7 shows the key delivery time for single-hop connection (*SH KEY TIME*), key delivery time for multi-hop connection (*MH KEY TIME*), and average key delivery time for all connections (*AVG TIME WEIGHTED*). *QKP* mode increases the *SH KEY TIME* of *BASIC* mode from 50 ms to 80 ms since *QKP* mode generates more keys than requested for future connections. However, since *QKP* relays multiple keys at once instead of relaying every key singularly as *BASIC*, *QKP* mode reduces the *MH KEY TIME* of *BASIC* mode from 240 ms to 140 ms. As *QKP* decreases the key delivery time of multi-hop connection, the *AVG TIME WEIGHTED* of *BASIC* is reduced from 160 ms to 110 ms.

In conclusion, the *QKP* mode has higher delivery times than the *BASIC* mode in the single-hop key requests, but lower times in the multi-hop requests, resulting in an improved overall performance.

D. Errors and Application Rejection Rate

The last simulation aims to find out the impact of the application admission control algorithm in proactively rejecting new applications if the QC rates are not sufficient, in order to avoid key errors to existing applications that are rekeying. The number of applications is set to 52, a relatively large number, such that the network is under high load and hence some applications must be rejected and key errors occur. We compare four scenarios: a *BASIC* mode and *QKP* mode without application rejection (named as *BASIC* and *QKP*, respectively), and *BASIC* mode and *QKP* mode with application rejection (named as *BASIC+REJ* and *QKP+REJ*, respectively).

Fig. 8 shows the percentages of applications rejected (*REJECTION*), percentages of applications with key errors (*CONN with ERR*), and percentages of total key errors (*KEY ERR*). As shown in Fig. 8, *QKP* reduces the *CONN with ERR* and *KEY ERR* from 25.4% to 4.4% and from 1.9% to 0.1%, respectively. This is because in the *BASIC* mode the first node in the path needs to generate two full keys, while in *QKP* mode the relaying key is utilized to relay multiple keys, hence the rate consumed by that key can be spread on those future keys. When application rejection is enabled, *BASIC+REJ* decreases the *CONN with ERR* and *KEY ERR* of *BASIC* from 25.4% to 16.6% and from 4.4% to 3.1%, respectively by rejecting 16.2% requests. In addition, *QKP+REJ* has no *KEY ERR* and *CONN with ERR* by rejecting 13.5% requests. We assume that this situation is preferable since users prefer their communications to be blocked at the beginning, rather than dropped because of a key error later on.

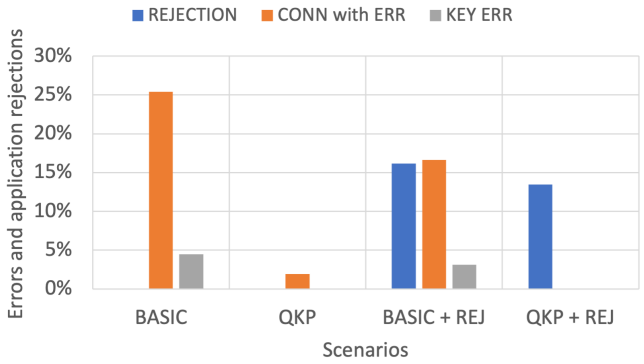

Fig. 8. Errors and application rejection rates.

From the simulations, we conclude that the *QKP* mode and the application admission control algorithm can both reduce key errors. Both the QKP mode and the application admission control algorithm have parameters that can be changed, even at run-time, to find an optimal trade-off between blocking and dropping.

VI. Conclusion

We implemented a prototype of a control architecture for QKD, that has been tested and evaluated by emulating the PoliQI network, posing particular attention on the Key Management Entity, to gain insights on how to implement, and possibly improve current QKD control solutions.

We developed a prototype of an SDN Controller and of a basic SD-QKD module, which can support single and multi-hop key distribution in any custom network topology. These components have been designed according to ETSI Standards, then some technical novelties have been introduced, such as i) introducing an algorithm to decide at runtime whether to reject a new connection and ii) developing a new procedure to establish end-to-end keys between non-adjacent nodes using Trusted Relay nodes and an SDN controller. Our experimental results demonstrate that these new functionalities allow to efficiently manage and distribute the keys to connections, compatibly with the limits imposed by the underlying QCs, while trying to minimize the errors. We plan to further reduce

the key errors by designing efficient QKD resource allocation algorithms.

REFERENCES

[1] A. Gatto, M. Brunero, M. Ferrari, *et al.*, "A BB84 QKD Field-Trial in the Turin Metropolitan Area," in *PSC*, 2021, Tu1A–2.

[2] A. Aguado, V. Lopez, D. Lopez, *et al.*, "The engineering of software-defined quantum key distribution networks," *IEEE Communications Magazine*, vol. 57, no. 7, pp. 20–26, 2019.

[3] T.-Y. Chen, X. Jiang, S.-B. Tang, *et al.*, "Implementation of a 46-node quantum metropolitan area network," *Npj Quantum Inf.*, vol. 7, no. 1, pp. 1–6, 2021.

[4] S. K. Joshi, D. Aktas, S. Wengerowsky, *et al.*, "A trusted node–free eight-user metropolitan quantum communication network," *Science advances*, vol. 6, no. 36, pp. 73–81, 2020.

[5] "Quantum key distribution (qkd); control interface for software defined networks," ETSI (European Telecommunications Standards Institute), Group Specification GS QKD 015 V2.1.1, Apr. 2022.

[6] M. Martinelli, P. Martelli, A. Gatto, *et al.*, "POLIQI: Milano quantum infrastructure," in *ICOP*, 2022.

[7] Q. Zhang, O. Ayoub, A. Gatto, *et al.*, "Joint routing, channel, and key-rate assignment for resource-efficient qkd networking," in *IEEE Globecom*, 2022.

[8] Y. Cao, Y. Zhao, Q. Wang, J. Zhang, S. X. Ng, and L. Hanzo, "The evolution of quantum key distribution networks: On the road to the qinternet," *IEEE Commun. Surv. Tutor.*, vol. 24, no. 2, pp. 839–894, 2022.

[9] Y. Cao, Y. Zhao, C. Colman-Meixner, X. Yu, and J. Zhang, "Key on demand (KoD) for software-defined optical networks secured by quantum key distribution (QKD)," *Opt. Express*, vol. 25, no. 22, pp. 26 453–26 467, Oct. 2017.

[10] N. Sala, "Implementation of a key management entity for quantum key distribution," M.S. thesis, Politecnico di Milano, 2022.

[11] "Quantum key distribution (qkd); protocol and data format of rest-based key delivery api," ETSI (European Telecommunications Standards Institute), Group Specification GS QKD 014 V1.1.1, Apr. 2019.

[12] P. J. Leach, R. Salz, and M. H. Mealling, *A Universally Unique IDentifier (UUID) URN Namespace*, RFC 4122, Jul. 2005.