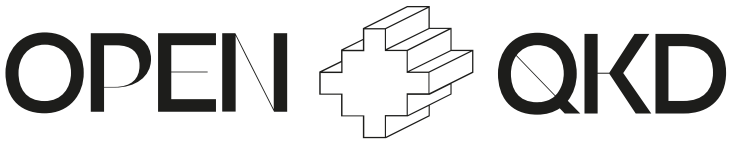
Call (part) identifier:	H2020-SU-ICT-2018-3
Topic:	SU-ICT-04-2019 Quantum Key Distribution Testbed
Grant Agreement / Contract Number:	857156
Project Acronym:	OPENQKD

Open European Quantum Key Distribution Testbed



Portal for virtual testbed access “QKXperience”

Deliverable: **D4.4**	Lead: VSB
Project month: M29	31.01.2022
Work package: WP4	Task: T4.3
Type: Other	Version: 1.0
Dissemination level: Public	

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857156.

More information available at https://openqkd.eu/.

Document Information

Author List

Organization	Name	E-mail
AIT	Florian Kutschera	Florian.kutschera@ait.ac.at
UNSA	Emir Dervisevic	emir.dervisevic@etf.unsa.ba
UNSA	Miralem Mehic	miralem.mehic@etf.unsa.ba
VSB	Ladislav Behan	ladislav.behan@vsb.cz
VSB	Lukas Orcik	lukas.orcik@vsb.cz
VSB	Miroslav Voznak	miroslav.voznak@vsb.cz

Reviewer List

Organization	Name	E-mail
OR	Jonathan Belhassen	jonathan.belhassen@orange.com
OEAW	Lukas Edengruber	Lukas.Edengruber@oeaw.ac.at

Version History

Version	Date	Reason / Change	Editor
0.1		Initial Release	Florian Kutschera
0.2	04.01.2022	Secondary Release	Miralem Mehic
0.3	14.01.2022	Third Release	Florian Kutschera
1.0	31.01.2022	Final Release	Florian Kutschera

Executive Summary

The aim of this deliverable is not only to provide an overview of the virtual test bed which displays all relevant information about the use case and its performance metrics collected from the use cases over the time of this project but also to provide an in-depth introduction of the test bed simulator.

Table of Contents

Abbreviations and Acronyms

This report uses the following abbreviations and acronyms:

QKD	Quantum Key Distribution
SDO	Standards Developing Organizations
QCI	Quantum Communication Infrastructure
SME	Small Medium Enterprise
API	Application Programming Interface
QRNG	Quantum Random Number Generator
SDN	Software Defined Networking
SDQN	Software Defined Quantum Networks
QT	Quantum Technologies
ETSI	European Telecommunications Standards Institute
ISO	International Organization for Standardization
TCO	Total Cost of Ownership
R&D	Research and Development
EU	European Union
CSA	Coordination and Support Action
TRL	Technological Readiness Level
PR	Public Relation
SW	Software
CA	Consortium Agreement
IA	Innovation Action
KPI	Key Performance Indicators
POC	Proof of Concept
MANET	Mobile Ad Hoc Networks
NFV	Network Function Virtualization
PoP	Point of Presence
ITS	Information Technology Solutions

1. Introduction

1.1. Purpose and scope of the document

The purpose of this document is to present the implementation of the virtual test bed that contains all performance data collected from the use cases on the one hand and also the test bed simulator, its function and some examples on the other hand.

1.2. Target audience

This deliverable is intended for a public audience as a how to guideline for navigating the virtual test bed as well as giving an overview of all the different settings of the test bed simulator that is interested in using these services.

1.3. Relation to other project work

Task 4.3, which this deliverable is based on, is connected to the work-packages 6, 7, 8 and 10.

Work-packages 6, 7 and 8 contain different parts of metrics collection from use cases and test beds that are accessible through the virtual test bed interface.

In work-package 10 the graphics produced by task 4.3 will be used for dissemination activities.

1.4. Structure of the report

This report will be divided into two parts.

The first part will describe the collection of performance data from the different test beds and will give an overview of the virtual test bed application itself which contains data about all use cases and performance data of the already running test beds that are providing data.

In the second part an in-depth description of the test bed simulator will be given. On the one hand it describes how the simulator is working and on the other hand to provide some basic understanding of its functions. Additionally, also some examples will be provided in order to give some possible testing scenarios which outcomes are explained in detail.

2. Data collection

2.1. Reference Collector

For collecting performance data from the different use cases the reference collector is used at various test beds of the OpenQKD project. The reference collector can either be deployed as a virtual machine or be installed on a physical computer. Either way, access to all devices that should be monitored is required. If it is not possible to access all devices from one collector node, multiple instances of the collector can be used in a single test bed.

In case the collector can not connect to the AIT server, for example because of a network problem, the collected data is cached for two weeks in the collector node before it is discarded.

The exact function of the reference collector is described in Deliverable 6.4.

2.2. Collector Scripts

For each type of device a unique script has to be created in order to collect the performance data. Those scripts can then be used by all test beds and use cases that are using the same device types. At this moment of time such scripts were already created for IDQuantique Cerberis3 and the Toshiba QKD Systems.

Those scripts have then to be adopted specifically for each test beds network architecture where they are used at and also for the exact data that is going to be collected.
That includes for example IP addresses, which data to collect and login credentials.

In the case of having devices from multiple vendors installed in the same test bed, collector scripts for each system type can be run on the same reference collector instance. It is only necessary to have multiple reference collector servers if not all systems can be reached from one system.

The collected data has then to be converted into the OpenQKD project specific message format. These messages, sent over Kafka, are sent together with a key. The key is used to describe the message: format, sender and a timestamp. For kafka the message is just a byte array with a maximum size of 1 MB. The key gives us the possibility to add more message formats in the future if necessary.

If the connection to the AIT can not be established, for example because of an internet outage, the data in the local Kafka is stored for two weeks but can be configured to buffer the data for even longer

3. Performance GUI

3.1. Database

The data sent to the central Kafka, from the local Kafkas at the test beds, is stored into an InfluxDB that is hosted at the AIT.

3.2. Visualization of Performance Data

The graphics used on the web interface are created by Grafana. These graphics are preconfigured specifically for each and every use case and shown datatype (see fig. 1).

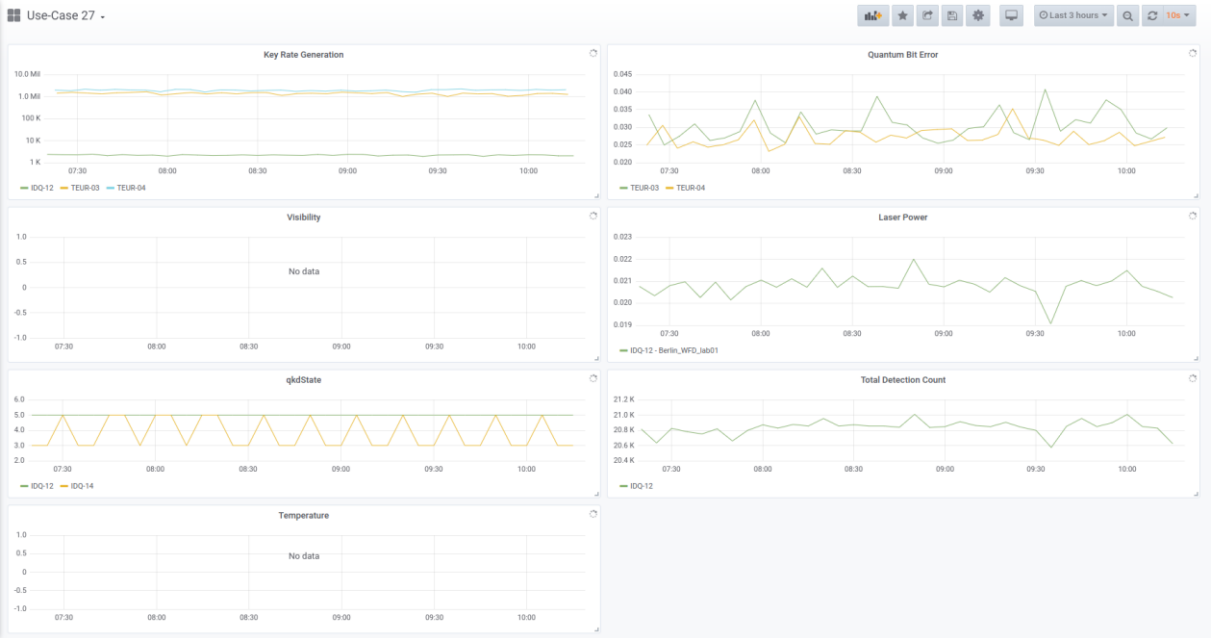


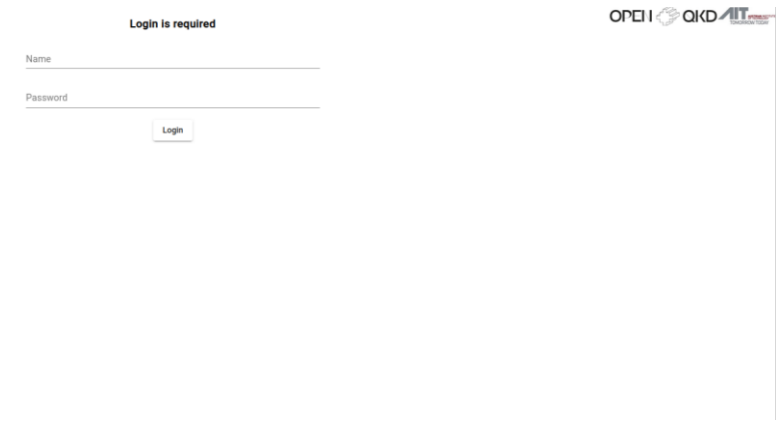
Figure 1: Preconfigured graphs in Grafana

Even though the graphics are predefined, the time range shown in these graphics can be changed freely through the time settings on the web interface. The duration and the time set will then be used for all shown use cases until it gets changed again.

The settings for the graphics are extensive and, among other things, the scaling of the axes and line thicknesses can be changed freely.

3.3. Web Interface

The access to the use case information, performance data and resulting graphics of the OpenQKD use cases is secured through user authentication and can be accessed over https://database-service.openqkd.eu/gui/ being the login page (fig. 2)



2: Login Page

Depending on the user that logs in, the data that can be seen might differ from what will be shown here.

After login the overview map is shown as indicated in fig. 3 where all use cases with basic information about them can be found. For easier distinction the use cases are color coded according to their current state. Black for upcoming, orange for ongoing and green if the use case has already concluded.

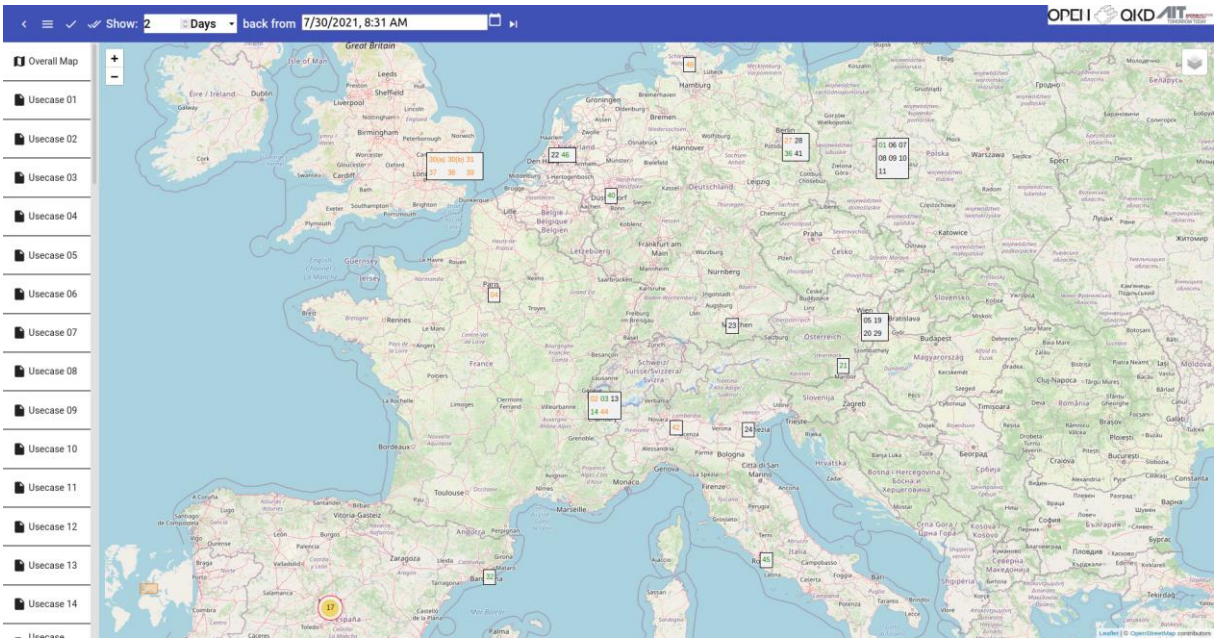


***Figure 3:** Overview Map*

After clicking on one of the use cases, depending on the amount of data received and user permissions, only the use case information or multiple options for more information about the use case, visualizations of the collected data or even an overview map that shows the different links that were used for that use case are shown.

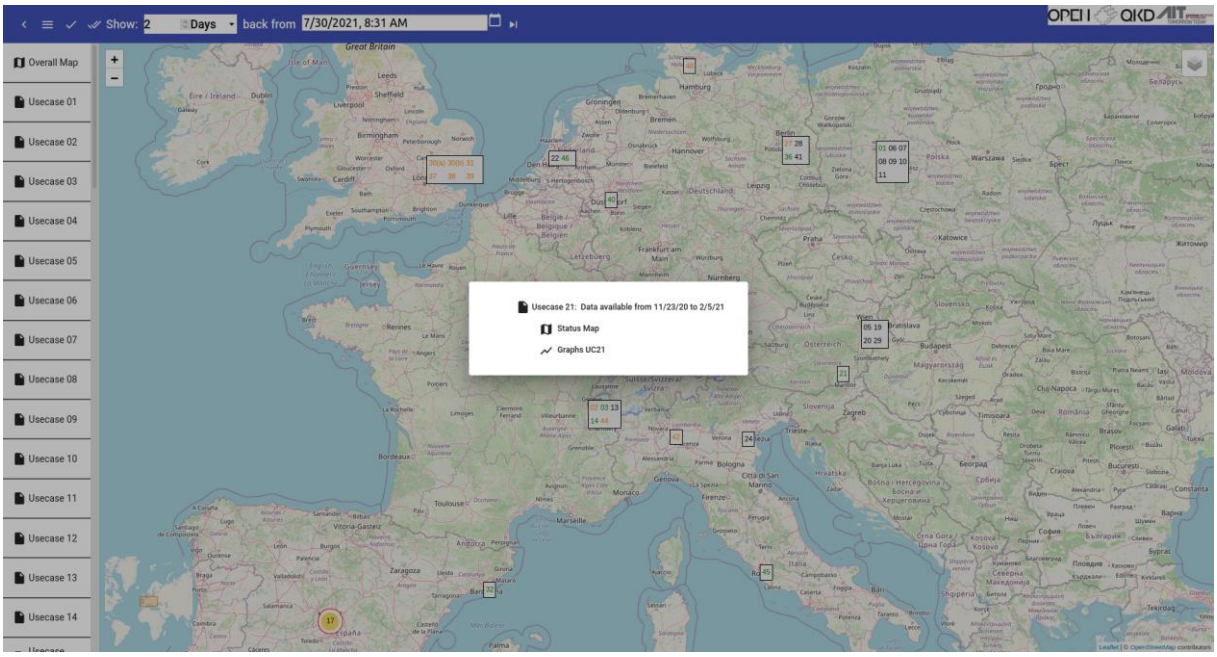


***Figure 4**: Overview Map Options*

If the logged in user has the necessary permissions, more information like a status map of the use case with additional information of the collected data, and also multiple graphics created from that data can be found. The time span from which information shall be shown can be selected freely. It is also possible to use the ⏭ button to jump to the point in time where the last data of a use case was recorded. If another use case is selected afterward, the “till” date won’t change automatically and it might be necessary to push the button again in case one of those use cases ended before the other.

In the top right-hand corner of the status map page is a list of metrics can be found that can be selected freely depending on what information should be displayed. The information displayed is an average value over the time that is selected on the top bar of the page.

If hovering over the lines or icons, information about the links and nodes will be displayed.

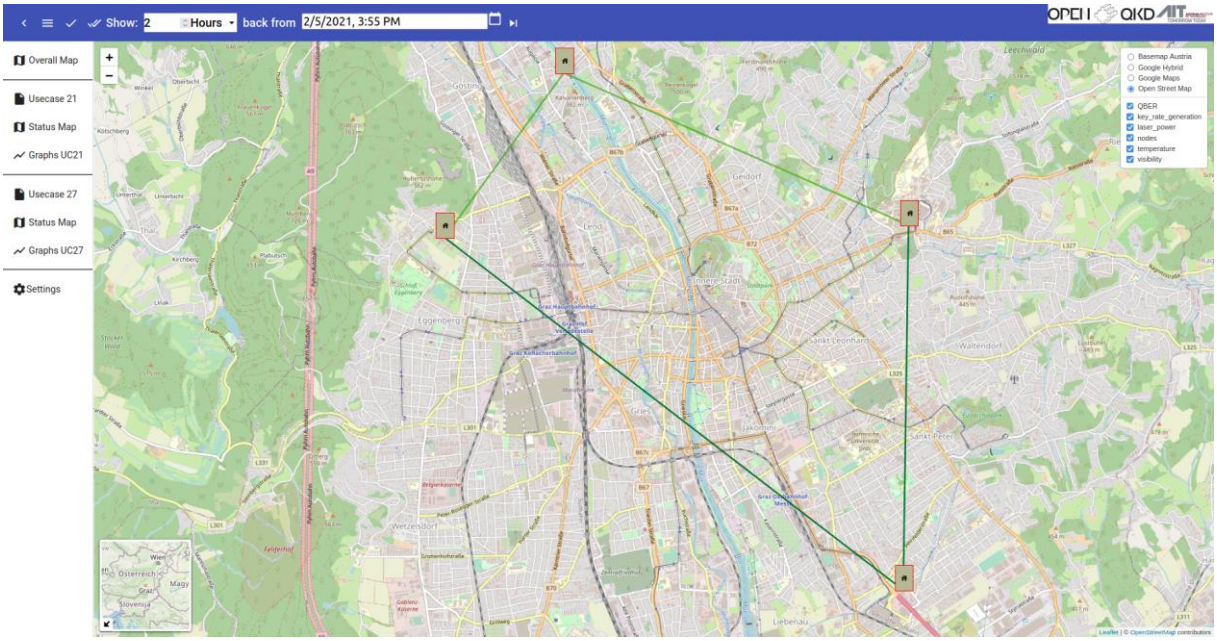


Figure 5: *Status Map*

On the Graphs page all performance metrics of the selected use case are shown each in their own plots. The plots shown might differ depending on the use case as not all use cases are collecting the same data-sets.

Figure 6: *Performance Data Graphic*

4. QKDNetSim web interface

The OPENQKD team has implemented a virtual web environment to simulate QKD links between remote locations accessible via link **www.open-qkd.eu**. It is a web interface for the quantum key distribution network simulator (QKDNetSim) being developed within WP6.

QKDNetSim is a simulation module designed to expand the NS-3 network simulator with QKD network functionalities [1]. Its primary purpose is to analyze different approaches to QKD network organizations, simulate networking technologies considering integrating QKD systems into existing telecommunications networks concerning network security [2]. The current stable version of QKDNetSim is 2.0.1, and it is compatible with the 3.33 version of the NS-3 simulator. During the QKDNetSim module development, no modifications were made to the core components of the NS-3 simulator (more details can be found in deliverable D6.3).

QKDNetSim follows the well-accepted organization of QKD node implementing components such as QKD key, QKD buffer (storage), QKD post-processing applications, and QKD encryptors [3]. Also, it is equipped with a Key Manager System (KMS) that supports ETSI GS QKD 014 [4] standard API interfaces for key delivery to the applications and implements functionalities that currently can support point-to-point communications. To verify KMS functionalities, dedicated QKD applications are developed, each employing a different API interface to communicate with the KMS. The QKD applications allow simulations of the QKD-enabled secure communications in various cryptography or network organization settings. All realized components are independently developed, enabling their installation on separate nodes and realizing large-scale network simulations using Message-Passing Interface (MPI) libraries on High-Performance Computing (HPC) platforms.

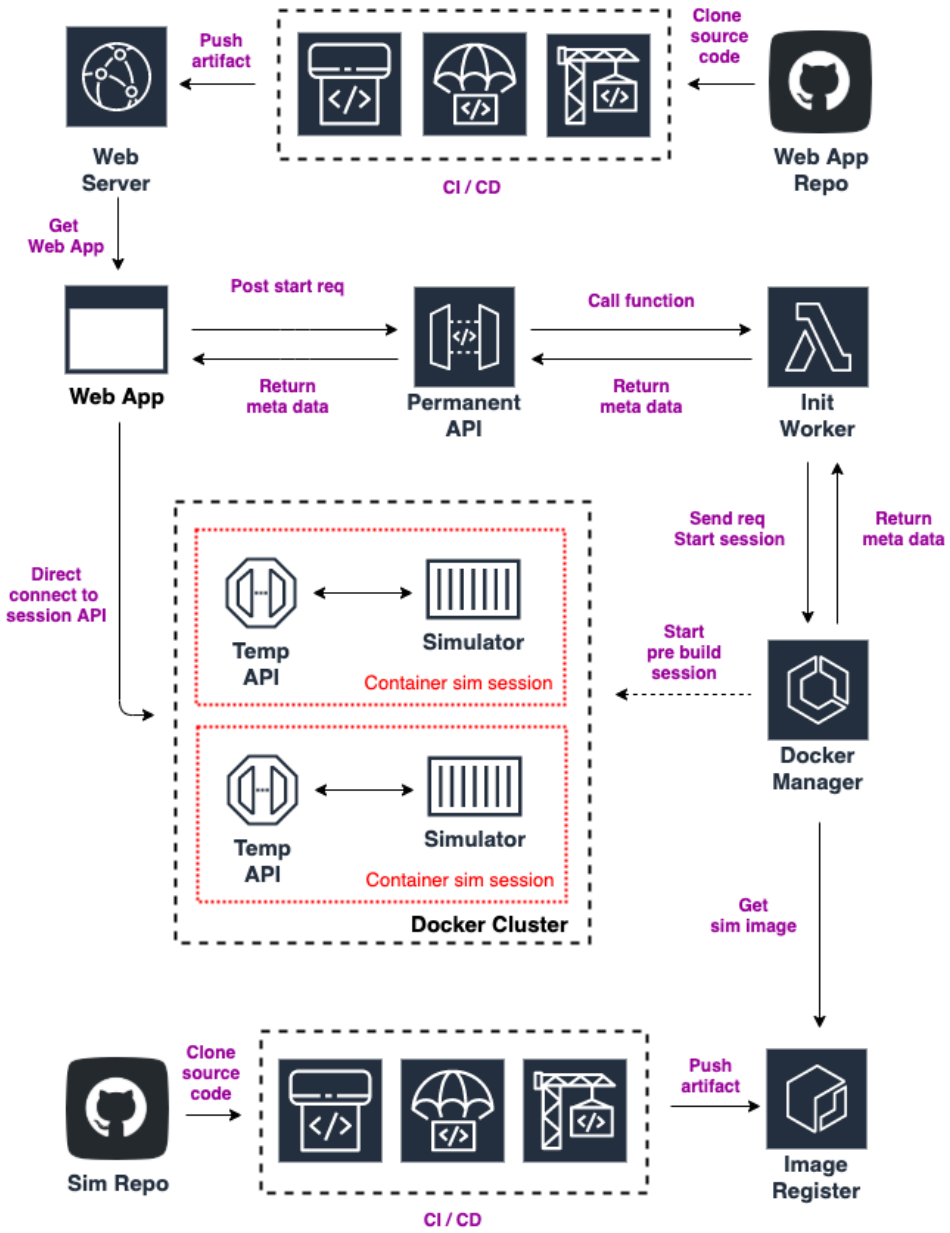


Figure 7: *QKDNetSim Web Interface Back-end Organization Scheme*

QKDNetSim is a console-oriented simulator that allows visualization of network topologies using the NS-3 tools NetAnim and PyViz. However, a web interface has been implemented for more accessible access to the network simulator (fig. 7). The web interface consists of two main parts, the graphical user interface (GUI) and the back-end. Through the web interface, the user can select points on the map, between which the distance is automatically calculated (based on the road distance between selected locations), and adjust simulation parameters such as the key generation rate, type of cryptographic techniques, and ETSI 014 parameters such as the number of keys requested from KMS entities in a single request, application and KMS operation time settings (fig. 8).

After entering all the parameters correctly, the application sends an API request to initialize the QKD simulator docker application. After completing the simulation process, the user is informed about the achieved results.

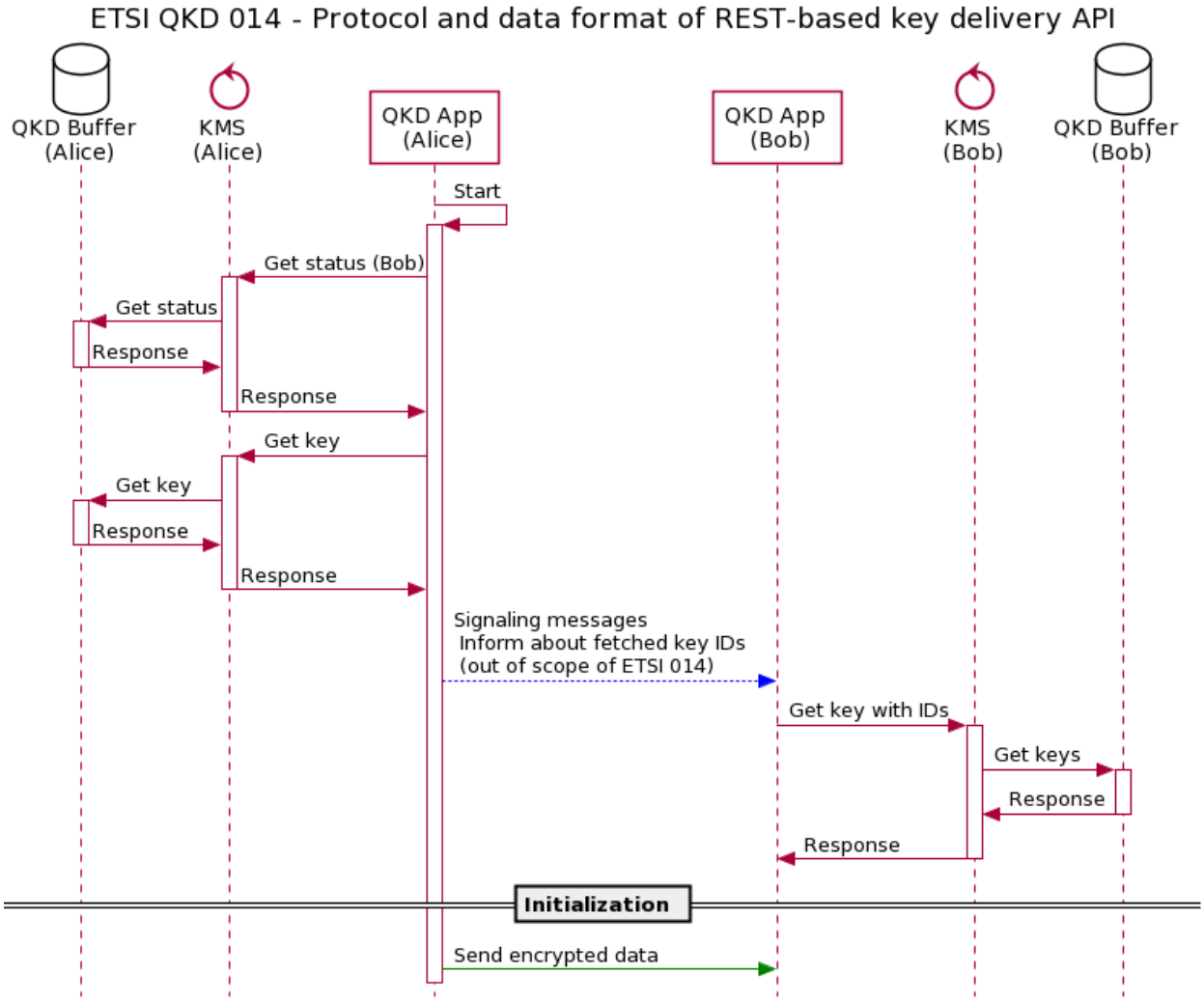


Figure 8: *The sequence diagram of communication between QKDNetSim entities based on ETSI 014 standard*

The simulation process begins with the user selecting locations of remote nodes on the map and defining simulation parameters, as shown in Figure 2. One of the fundamental parameters for setting is the QKD key rate, which is the intensity of QKD key generation. Given that today's QKD systems are limited to a range of up to about 50 - 80 km where the QKD key rate is 10-50 kbps [2], the user can define a smaller distance to increase the key rate. However, for distances longer than 80 km, it is necessary to use a trusted relay approach in which the total intensity of the QKD path is limited by a bottleneck link (approximately 10 kbps). Additionally, the user defines the parameters of the post-processing application by specifying the average intensity and size of the post-processing traffic packets (fig. 9). The increased post-processing traffic can affect the key generation rate and the speed of sending user data traffic due to the risk of congestion on public channels [5]. Then, the user defines the data traffic settings. Namely, the types of cryptographic algorithms used, the size of data packets, and traffic intensity are determined.

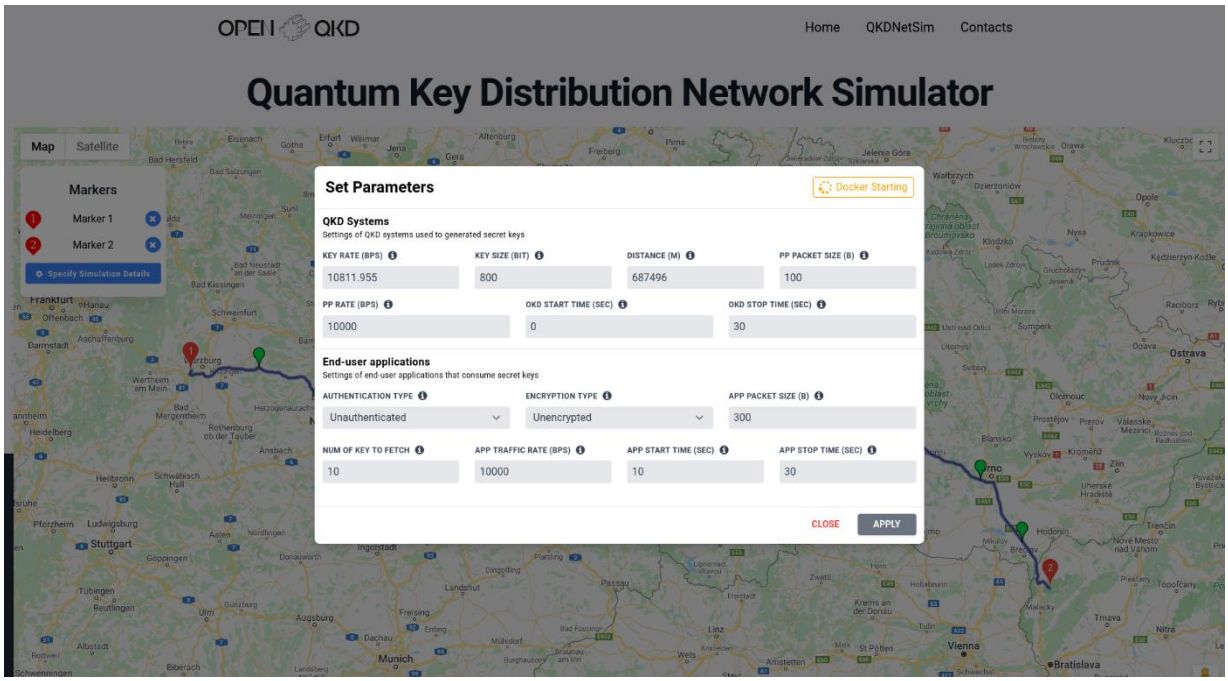


Figure 9: QKDNetSim web interface parameter configuration

The parameter that defines the number of packets served by the KMS according to the application is of particular interest. Suppose the KMS does not have enough keys in the warehouse. In that case, it is not possible to deliver keys to the application, and it is not possible to exchange encrypted traffic between remote applications. The deployed version of QKDNetSim does not terminate if there are not enough keys for traffic protection. That is, only those packets for which there are enough keys will be protected. Accordingly, a parameter that defines the number of packets that will be encrypted if the AES algorithm is selected is also of interest (see example #2).

Once the Docker instance has been successfully allocated (the icon in the upper right corner turns green), the user can run the simulation and analyze the collected data (fig. 10).

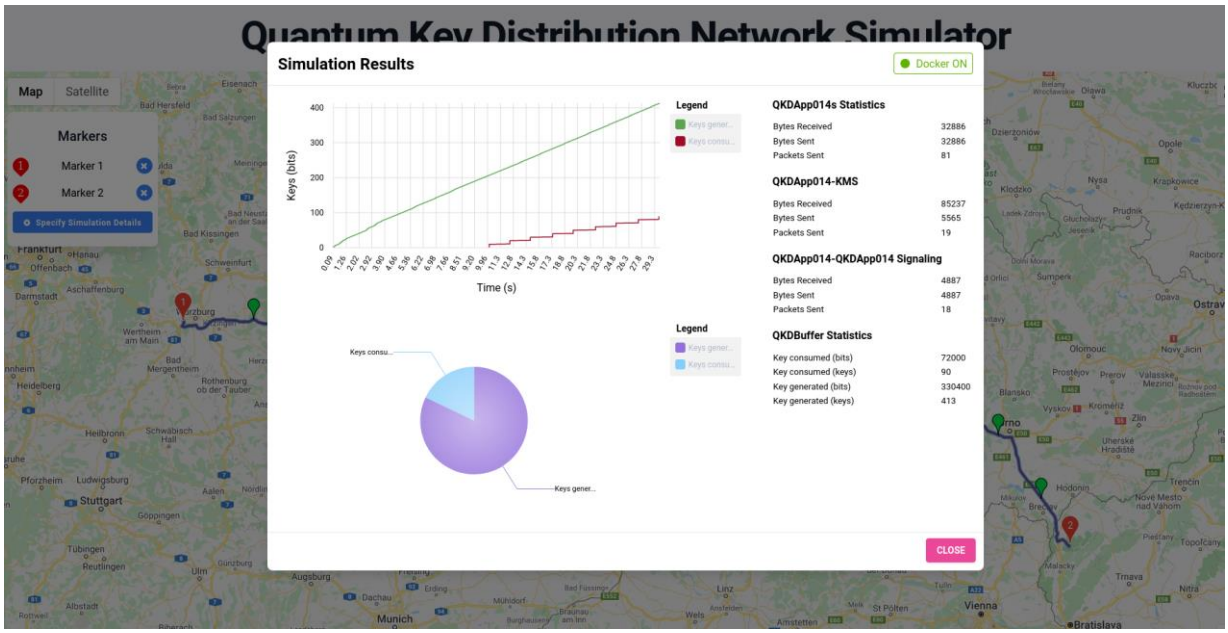


Figure 4: The output results of the simulation

5. Examples

This section provides an overview of the simple usage of QKDNetSim with the integration of the ETSI 014 standard through examples. Throughout this section, key information is provided to the user to assist in correctly interpreting results. QKDNetSim allows communication between two applications running on adjacent QKD nodes. To avoid confusion, the web interface allows users to adjust the settings of QKD applications that are in charge of establishing keys. In the rest of the text, these applications are denoted as QKD post-processing applications while key-consuming applications are simply denoted as Alice's and Bob's applications.

Let's assume Alice's application wants to communicate with Bob's application securely. In that case, Alice's application will apply some cryptographic methods to the sensitive data to make sure it's secure transit to its recipient, Bob's application. The application can choose between the OTP (One-time pad) and the AES-128 encryption methods to assure confidentiality.

Moreover, if the application wants authentication services for its traffic, VMAC and SHA1 are available options. The encryption (and the VMAC authentication) requires symmetric secret keys to be established between Alice's and Bob's applications. In QKDNetSim, applications are served by KMS with QKD key material as symmetric keys, and the communication between them is defined by ETSI GS QKD 014 standard. The users can define not only cryptographic methods and specific parameters related to them (such as the AES lifetime) but also the application's data rate and packet size, for example. Furthermore, some key parameters of the QKD layer are also specified by users' desires, as explained in the previous section. This section provides examples, e.g., experiments and results analysis.

NOTE: *If the application desires OTP encryption, the packet size (defined in bytes) must be less or equal to the default size of the QKD keys generated and stored in QKD Buffers on the KMS. The default size of the QKD key is set to 8092 bits and could be changed by the user. This limitation comes from the fact that KMS application deployed in the current version of QKDNetSim web simulator does not support key management functions which would allow shaping (key merge and split operations as well as the other KMS functionalities that will be available in the next version of QKDNetSim) of the QKD keys, and thus, the KMS can provide only keys whose size is less than 8092 bits.*

5.1. Example #1

This first example provides an application's performance analysis and quantum key usage against different cryptographic methods. The constant parameters in these simulations are listed in Table 1. The obtained results of this experiment are shown in Table 2.

Table 1. Constant simulation parameters. It is recommended to set the earlier start time of the QKD post-processing application in relation to the start time of the application that consumes the keys, so that the keys in the QKD buffers can be generated in advance.

	Parameter	Value
Application configuration	Start time [s]	10
	Stop time [s]	50
	Number of keys to fetch per *get_key* request	10
	Data rate [kbps]	10
	Packet size [bytes]	300
QKD configuration	Start time [s]	0
	Stop time [s]	50
	Key rate [kbps]	100

Some specific parameters regarding the KMS configuration are not available for users to configure through the web virtual environment. They can be specified in the source code. However, the users of the simulator should be aware of them! For example, the KMS is configured to allow ETSI 014 GET_KEY request if the requested number of keys is no more than 100. Furthermore, the requested key size must be multiple of 8 (this will be satisfied as the user, if he uses OTP, specifies packet size in bytes, thus, key size will always be multiple of 8). If some of the users' parameters do not adhere to the KMS rules, the KMS will respond with an error message which will provide users with the information of the error, and the simulation will be stopped. However, the error returned because the KMS currently does not have enough key material to serve the application will not stop the simulation, rather, the application will simply try again after a constant amount of time has passed (3 seconds).

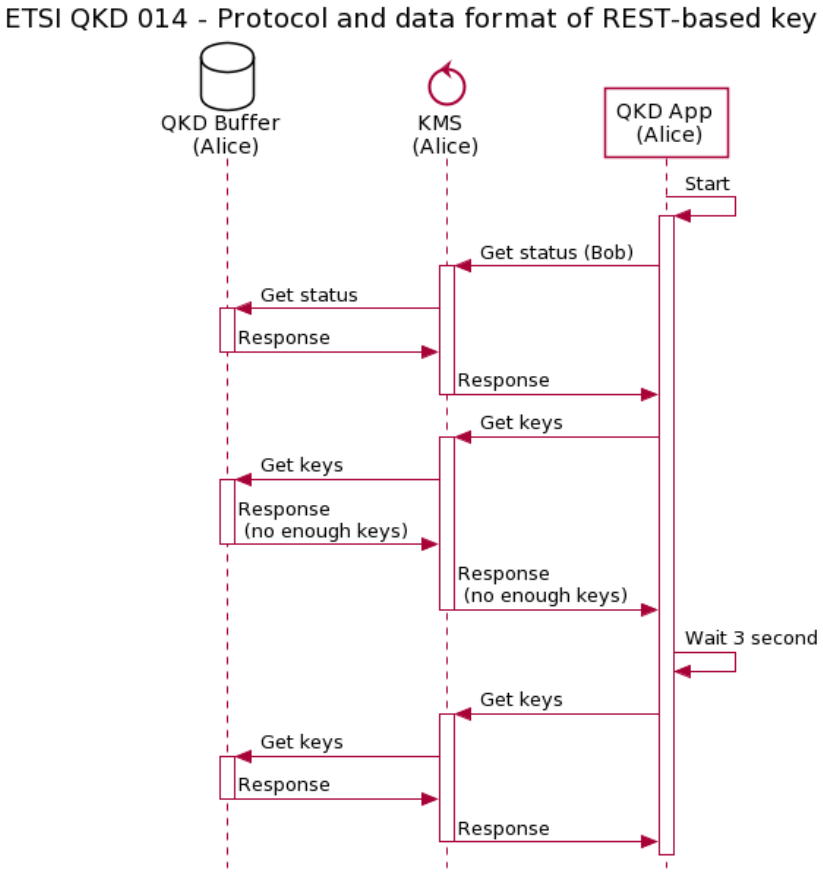


Figure 5. If not enough keys are available, the QKD Application will wait 3 seconds before requesting keys from KMS again. Users can specify how many keys the application will request from KMS in its requests.

Table 2. The comparison of exchanged data and signalling packets between QKDApps and QKDApps and KMS in case of different cryptographic methods being employed

Cryptographic method	# data packets exchanged	# signaling data packets exchanged	# packets to KMS exchanged	# keys consumed	# keys generated
Unprotected	167	0	0	0	586
OTP	161	34	35	170	586
OTP + VMAC	60	13	15	70	586
AES (*refresh 5)	164	8	9	40	586
AES (*refresh 5) + VMAC	166	8	9	40	586
AES (*refresh 10)	167	4	5	20	586
AES (*refresh 10) + VMAC	167	4	5	20	586

* The current version of QKDNetSim specifies the AES refresh rate in the number of packets *that are encrypted using the same secret key!*

Signaling data packets are packets through which Alice's application notifies Bob's application of the obtained key material conveying their keyIDs. Based on this information, Bob's application will use ETSI 014 GET_KEY_WITH_KEYIDS method to obtain (or try to obtain, as this process could fail if some of the requested keys are not available or are already reserved for other purposes) the same set of quantum keys. Alice's application cannot use the obtained key material until Bob's application sends confirmation that it also successfully obtained the requested set of keys. Signaling messages are implemented using REST API, and their specification is out of the scope of the ETSI 014 standard.

Column "# keys consumed" registers the number of keys consumed from the KMS perspective. In the case of OTP, users expect that the number of packets application sent should be the same as the number of keys consumed (as each packet is encrypted with a new key), but results show otherwise. This can be explained as follows: the application obtains the set of keys (in this case 10 keys) per request, and starts using the keys. Due to simulation time, the application did not use all keys (from a set of 10 keys) obtained from the KMS, but from the KMS perspective, these keys are consumed as they are served to the application. Therefore, the number of packets sent by the application in the case of OTP can be lower than the keys consumed -- from the KMS perspective -- (but not lower than the size of the set, which is 10 in this case).

The number of keys consumed when encryption and VMAC authentication are applied compared to the case where only encryption is applied is unchanged. This is due to the current application design. If the application desires to use encryption and VMAC authentication, it will obtain keys whose length is encryption_key_size + authentication_key_size. Later, the application will use the first portion of the key for encryption purposes and the second for authentication.

The results show that the impact on the application's performance, that is, its data rate, is negligible in every scenario (excluding OTP+VMAC). As expected, in the case of AES encryption, keys are requested less frequently. Moreover, one should keep in mind that these keys are only 128 bits long (not only the number of keys consumed is lower compared to the OTP, but the actual key consumption if compared in bits is significantly lower). The VMAC authentication should not have much of an impact on the application's performance, and it is correctly shown in the case where AES encryption is combined with VMAC. However, the application's data rate (determined by the number of packets sent) is significantly lower than expected if OTP is combined with VMAC. This is due to an unexpected increase in delay between application and KMS, which is introduced in some cases due to fragmentation on the IP layer and the inability to send small fragments immediately (TCP Silly Window Syndrome avoiding algorithms). Excluding this case, it is shown that the latency introduced to obtain QKD keys from the KMS has no impact on the desired data rate of the experiment.

Key consumption given in bits shown in the results segment is much higher than users expect. However, as it is previously mentioned, the KMS does not support key split or merge functionality, and thus, the KMS prepares a QKD key (whose size is 8092 bits) in a key whose size is one requested by the application (128 bits in case of the AES encryption). The rest of the QKD key is then deleted, resulting in a significant waste of the QKD key material! The KMS records this waste of the QKD key material in "keys consumed in bits" together with the portion of the key used.

5.2. Example #2

Example #1 showed that the latency introduced in obtaining the key material for a 10 kbps application's data rate is not enough to disrupt the application traffic. Example #2 shows that the latency can impact the achieved data rate if the desired data rate is higher than 10 kbps. Moreover, it shows the effect of the parameter "number of keys to fetch per request", as key requests can be more or less frequent. The results for different application data rates (but also encryption methods) are shown in Table 3. The parameters not mentioned here are the ones used in example #1.

Table 3. Performance analysis for various application's data rate.

User defined application data rate [kbps]	Encryption method	# data packets exchanged	# keys consumed
10	Unprotected	167	0
	OTP	161	170
	AES (10)	167	20
20	Unprotected	334	0
	OTP	280	280
	AES (10)	328	40
50	Unprotected	834	0
	OTP	130	140
	AES (10)	795	80
100	Unprotected	1667	0
	OTP	140	150
	AES (1)	270	270
	AES (10)	1486	150

The results show that the achieved application data rate noticeably decreases in the case of OTP cipher being employed. If the application data rate is 100 kbps, the application transmitted 1667 packets when no protection is applied. Still, if the OTP cipher is used, the number of transmitted packets significantly decreases to 140.

Two design implementation details, as well as the network performances, influence the application performance. The first one is that the application is relatively simple: it obtains the desired number of keys via one *GET_KEY* request from the KMS and starts sending the data. However, when Alice's application consumes the acquired number of keys, data traffic is stopped until a new set of keys is obtained. The overall latency (which includes latency for Alice's application to obtain a new set of keys, signaling between Alice's and Bob's application as well as the latency for Bob's application to obtain the same set of keys from its local KMS) is influenced by the number of keys KMS has to provide via single response, and the size of the keys (the performance of the link though which signaling data is sent can also be considered). If the number of keys with given sizes KMS needs to return is large, the application will wait longer for the response due to the transmission delay. This impact is best shown by comparing the number of transmitted packets when OTP and AES (1 - refresh rate) is employed for the desired data rate of 100 kbps.

The second thing to consider is the performance of the QKD layer. If the application's request cannot be satisfied (the requested number of keys is not available), the application is stopped. It will periodically probe KMS (every 3 seconds) to obtain a new key material if it becomes available. If the desired application's data rate is too high and the user wants to use OTP or even AES (1 - refresh rate), the KMS will run out of keys sooner, and on those occasions, the application will wait 3 seconds before trying again. KMS must have enough keys available to respond to the application's request (parameter "number of keys to fetch per request"). This will significantly limit the application data rate, and the results obtained in the case of AES (1) and AES (10) represent this scenario. The use of AES (10) will result in less frequent requests to the KMS, and thus, the situations where KMS does not have the available number of keys to respond to the application are more likely avoided.

The number of keys to fetch per request can also impact the performance. As mentioned previously, the large number of keys KMS has to return in a single response can increase the delay between application and KMS due to large amounts of data to transmit, especially if the size of the keys is also significant as in the case of OTP cipher. Moreover, while the application can request a large number of keys and thus, make less frequent requests to the KMS, it is more likely that the KMS will not be able to satisfy these application requests, resulting in a more frequent application "punishment" of 3 seconds in waiting before making a new request. The small number of keys, on the other hand, can result in frequent key requests, which can quickly increase the time where the application waits for new key material.

While it seems less possible for KMS to run out of keys in some of the presented cases, due to the lack of the key merge and split functionalities at the KMS, the QKD key buffer empties more quickly (as a large amount of the quantum key material is wasted).

6. Summary

7. References

[1] M. Mehic *et al.*, “Implementation of Quantum Key Distribution Network Simulation Module in the Network Simulator NS-3,” *Quantum Inf. Process.*, vol. 16, no. 10, p. 253, Oct. 2017, doi: 10.1007/s11128-017-1702-z.

[2] M. Mehic *et al.*, “Quantum Key Distribution: A Networking Perspective,” *ACM Comput. Surv.*, vol. 53, no. 5, 2020, doi: 10.1145/3402192.

[3] V. Martin *et al.*, *Quantum Technologies in the Telecommunications Industry*. The Author(s), 2021.

[4] ETSI ISG QKD 014, “Quantum Key Distribution (QKD); Protocol and data format of REST-based key delivery API,” vol. 1, pp. 1–22, 2019, [Online]. Available: https://www.etsi.org/deliver/etsi_gs/QKD/001_099/014/01.01.01_60/gs_qkd014v010101p.pdf.

[5] M. Mehic, O. Maurhart, S. Rass, D. Komosny, F. Rezac, and M. Voznak, “Analysis of the Public Channel of Quantum Key Distribution Link,” *IEEE J. Quantum Electron.*, vol. 53, no. 5, pp. 1–8, Oct. 2017, doi: 10.1109/JQE.2017.2740426.