Universitá Ca' Foscari Venezia

Master Degree in
Computer Science

Final Thesis

Quantum Key Distribution Performance: Extending the SeQUeNCe simulator to Estimate the Speed of free-space Optic Links

Supervisor
Prof. Leonardo Maccari

Graduand
Luca Vinci
868166

Academic Year
2022/2023

Contents

Chapter 1

Introduction

Making our communication secure over untrusted networks is a crucial aspect of cryptography. Currently, the common approach involves using Public Key Cryptography, which rely on complex math problems and assumptions about the computational power of potential attackers. In theory, these methods fall into the category of security measures that could be broken with enough computational power. With the advancement of technology and the rise of quantum computing algorithms, like Shor's Algorithm [2], that can efficiently solve certain complex math problems, our current security solutions face significant threats and might not be reliable in the long run.

Quantum Key Distribution (QKD)[5] uses quantum information principles to make secure cryptographic keys. Unlike regular keys, these quantum keys aren't restricted by the increasing computational power of computers. By using quantum properties, QKD is a safe way to share keys.

The goal of the thesis is to implement a Quantum Key Distribution (QKD) network where a message exchange from node A and node B is possible only on every hop on the path, made on Free Space Optic (FSO) links, connecting the two nodes has a quantum key generated with the Bennett-Brassard (BB84) [7] and Cascade [25] protocol and subsequently analyze its performance. To do so it was used an open source quantum simulator written in Python called SeQUeNCe (Simulator of Quantum Network Communication) developed by Argonne National Laboratory [1].

In the extension we assume that the nodes of the network, where we will run the simulations, are trusted nodes. This is because communication happens hop by hop, and each node decrypts the traffic before encrypting it again. Through SeQUeNCe, we modeled an extension for simulations on trusted QKD networks, representing the quantum channel as a Free-Space Optical (FSO) link. Parts of the SeQueNCe simulator's source code were modified to make the extension effective and accurate. We will discuss the various changes in the chapters related to the implementation.

We noticed during the development of the extension that SeQUeNCe faces some memory management issues. This becomes apparent, especially when running protocols for Quantum

Key Distribution (QKD) and correcting errors in the generated keys. In fact, simulations under heavy load could end up using all available memory during their execution. For instance, simulations on networks with 15 nodes or more, under heavy load, and with a simulation time of 1 second, in our tests, reached consuming more than 20 GB of RAM. We therefore note that as the execution time of the simulation increases, the memory increases linearly with time, thus reaching a point where it is completely saturated. During the project an issue [1] was opened on the SeQUeNCe GitHub page to make it known to developers.

While writing the thesis I made use of translation tools to improve the English form, tools such as Google Translate and ChatGPT. Tools used for the sole purpose of translation.

[1]Issue link: https://github.com/sequence-toolbox/SeQUeNCe/issues/174

Chapter 2

Theoretical Background

The fundamental idea behind QKD is the use of two main principles: the Heisenberg uncertainty principle and the no-cloning theorem to securely exchange keys between Alice (the sender) and Bob (the receiver). The Heisenberg uncertainty principle states that it is impossible to measure certain properties of a quantum particle (such as speed and momentum) with unlimited precision. The other principle that composes the basis of QKD is the no-cloning theorem. It states it is impossible to create an identical copy of an unknown quantum state.

As mentioned earlier, with the rise of quantum computers, the security of our communications using traditional cryptography might not be enough. That's where quantum cryptography becomes important. At the core of it, we rely on a significant result from quantum physics, Heisenberg's Uncertainty Principle, along with Quantum Entanglement. Before going into the details of Quantum Key Distribution (QKD), it's essential to introduce some basic concepts. So, in this chapter, we'll get to know these fundamental ideas.

2.1 Definition of a Qbit

Quantum communications uses what is called Qbit. A Qbit is an analogue to the classical bit. In addition to having a value of 0 or 1 it can be in a so-called superposition of the two. This is due to the principles of quantum mechanics, which allow for particular phenomena such as superposition and entanglement. Thanks to this ability, qubits have the potential to perform calculations in parallel, making a so-called quantum computer faster than a classical one. Qubits can be made using various physical systems, such as photons.

So, what's interesting for communication purposes is that we can encode information on top of it. So a photon, for example, can be seen as a Qbit. We can encode information inside with plarization. In technical terms, polarization refers to the orientation of the vibration of an electromagnetic wave, such as a photon. The polarization can be horizontal, vertical, or in any of the infinite possible angular directions between these two. They are usually polarized into one of two orthogonal states, for example, horizontal or vertical. The quantum information is therefore encoded in the polarization of the photon. An example of polarization states is the following:

- You can associate the horizontal polarization, written in this form $|H\rangle$ called Dirac notation, with bit 0 and the vertical polarization $|V\rangle$ with bit 1.

The photon, or Qbit, will subsequently be sent to a destination through a so-called quantum channel, for example an optical fiber, at the destination. The measurement depends on the state of the photon and the direction of the analyzer. For example, if a photon is vertically polarized and the analyzer is set to measure horizontal polarization, the probability of detecting the photon will be zero. It is important to note that reading a photon will change its state.

2.2 Entanglement

Entangled states can involve two or more qubits. In entangled states, the individual states of each qubit cannot be described in isolation. In other words, when the wave function of one qubit collapses due to measurement, it can instantaneously influence the state of the other, regardless of the physical distance between them.

To know what entanglement is, we just need to see how it's not like independent or classically connected systems. Systems are independent when knowing about one doesn't really tell you anything about the other [4]. For example, suppose we have two systems that are two objects. Our objects come in two shapes, square and circular, which we identify as states. Then the four possible joint states are (square, square), (square, circle), (circle, square), (circle, circle).

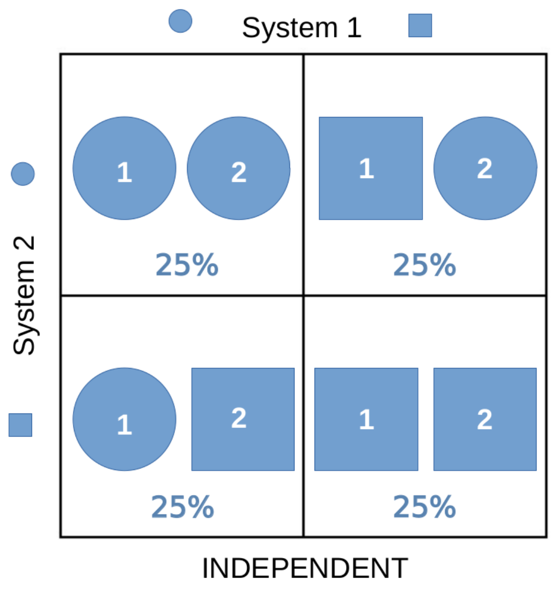

Figure 2.1: Example of what the odds might be of finding the system in each of these four states.

We say that our objects are ”independent” if knowing the state of one of them does not provide useful information about the state of the other. The table has this property. If the first object is square, we are still unaware of the shape of the second. Likewise, the shape of the second reveals nothing useful about the shape of the first.

If our objects are entangled, information about one also reveals something about the other. In that case, whenever the first object is circular, we know that the second is also circular. And when the first is square, so is the second. Knowing the shape of one, we can definitely deduce the shape of the other.

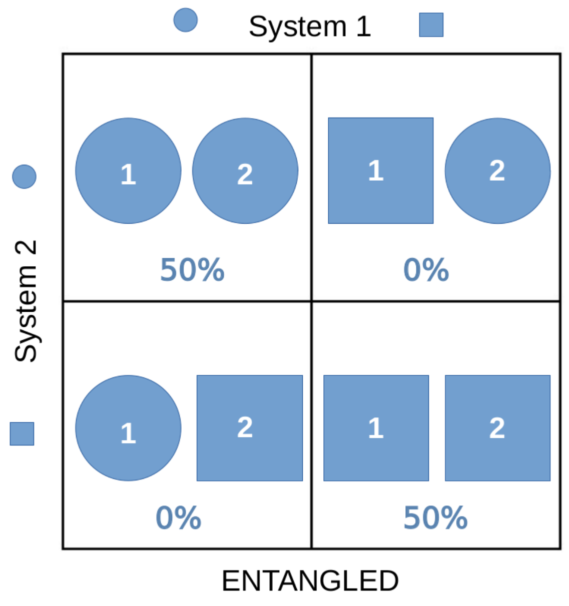

Figure 2.2: Example of what the odds might be of finding the system in each of these four states.

The quantum version of entanglement is essentially the same phenomenon, namely the lack of independence. In quantum theory, states are described by mathematical objects called wave functions.

The two-level system of a quantum computer can be represented by the vertical and horizontal polarization of a photon, the spin up and spin down of an electron, or any other proposed state variable [1]. Two or more Qbits can be in an entangled state. When states are entangled, the state of each QUbit cannot be described independently. That is, collapsing the wave function of a QUbit by measurement can immediately affect the other state of the pair regardless of the physical distance between them. Consider someone who prepares two photons whose polarization is intertwined and then sends them in opposite directions along a fiber-optic cable. Classically, when we measure that one photon is polarized vertically or horizontally, we know for sure that the other will be the opposite.

Chapter 3

Quantum Key Distribution

3.1 Cryptography Overview

In recent years, quantum cryptography has seen significant attention and rapid progress [5]. The central challenge of cryptography in this period was the secure distribution of keys to authorized users, ensuring the confidentiality of these keys in the face of potential adversaries. This challenge is commonly called *secret key agreement problem.*

A cryptosystem achieves information-theoretical security (ITS), also called unconditional security[24], when its security **relies solely** on principles from information theory. In other words, it does **not rely** on unverified assumptions regarding the complexity of certain mathematical problems, and therefore remains secure even in the presence of adversaries with unlimited computational capabilities.

3.1.1 Computationally secure symmetric-key cryptographic schemes

Symmetric-key cryptography, a cryptographic approach where both the sender and receiver utilize a shared secret key, is employed to ensure the confidentiality of encrypted messages. Secret key agreement, based on classical and computationally secure symmetric-key cryptography, can be realized by exclusively utilizing symmetric-key cryptographic primitives. For instance, combining a symmetric-key encryption scheme with a symmetric-key authentication scheme enables the creation of a secret key agreement primitive.

The security of confidential key agreement in traditional symmetric-key cryptography depends on the security of the cryptographic building blocks used and how they can be combined. Claude Shannon demonstrated that there's no encryption method that's unconditionally secure and requires fewer key bits than the one-time pad[23][5]. One-time pad (OTP) is a cipher that uses a unique secret key, which is the same length as the message to be encrypted, and it is used only once. The secret key is made up of a random sequence of bits or characters. To encrypt a message, each bit of the message is combined bit by bit with the corresponding bit of the secret key, using a logical operation like XOR . The result is the encrypted message. To decrypt, the same process is used, combining the encrypted message with the secret key. This bears a fundamental implication: to establish an unconditionally secure scheme, the entropy of the encryption key must be at least as extensive as the message to be encrypted [5].

Nevertheless, it is possible to construct a secret key agreement scheme using classical symmetric-key encryption and authentication schemes that lack unconditional security. The security model applicable to such symmetric-key classical encryption schemes does not meet the criteria of unconditional security, as the key's entropy is smaller than that of the message, nor does it offer provably computational security.

3.1.2 Quantum key distribution

Quantum key distribution (QKD) [7] emerges as a quantum cryptographic remedy for the challenge of establishing a secret key agreement between two mutually trusting users in the presence of potential adversaries, offering an alternative to conventional public-key cryptography. Unlike its counterparts, QKD is established as unconditionally secure, guaranteeing protection regardless of the computational capabilities an attacker may possess.

To understand the fundamental structure of QKD, consider a QKD link. Illustrated in the figure 3.1, a QKD link forms a direct connection between two users, Alice and Bob, who aim to exchange secret keys. The QKD link include a quantum channel and a classical channel. Alice generates a random stream of bits, encoding them into a sequence of quantum states of light transmitted through the quantum channel. Bob, upon receiving these quantum states, conducts measurements that yield classical data correlated with Alice's bit stream. The classical channel is then employed to verify these correlations [5]. If the correlations meet a certain threshold, it statistically indicates minimal eavesdropping on the quantum channel. Consequently, there is a very high probability of distilling a perfectly secure symmetric key from the correlated data shared by Alice and Bob. Contrary, if the correlations fall below the threshold, the key generation process is halted and restarted. Particularly, any substantial disturbance on the quantum channel, causing noise exceeding the security threshold, practically disrupts key generation. An active attacker with access to the quantum channel can effectively launch denial-of-service (DoS) attacks in such cases [5].

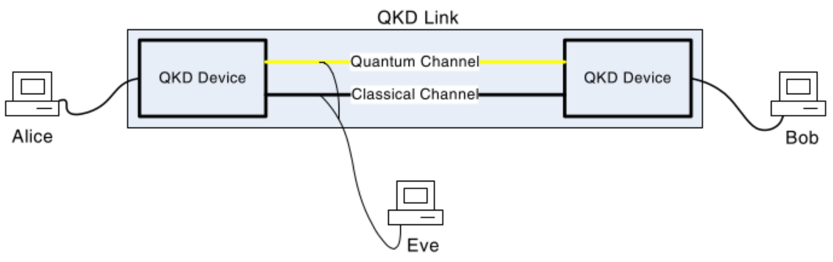

Figure 3.1: Structure of a QKD link. Figure taken from [5].

Ensuring the authenticity of the classical channel and maintaining constant security in Quantum Key Distribution (QKD) requires the use of a symmetric secret key agreement method [5]. This approach involves having initial resources, specifically a public quantum channel and an authenticated public classical channel. To adhere to the highest security model, known as the Information-Theoretically Secure (ITS) paradigm, message authentication codes based on universal hashing can be employed to authenticate communications on the classical channel. To incorporate ITS authentication into QKD, Alice and Bob need to share a short secret key beforehand. When QKD operates within this framework, it acts as an Information-Theoretically Secure symmetric secret key expansion scheme [5].

Enhancing the security of a direct classical communication link involves integrating Quantum Key Distribution (QKD) with symmetric encryption. QKD serves as a secret key agreement method achievable at the physical layer. Now, our focus shifts to exploring how the secret keys generated through QKD can be employed for a link layer cryptographic objective, specifically, safeguarding the transmitted data on a classical communication link. This is achieved by leveraging keys produced by QKD, in conjunction with initially shared compact secret authentication keys, and employing symmetric-key cryptographic primitives.

In a more formal context, we address the challenge of transmitting classical messages (payload) securely from Alice to Bob through the following generic protocol [5]:

1. Establishment of a symmetric secret key $K_S = K_{encrypt} \cdot K_{auth}$ between Alice and Bob. Where $\cdot$ is a concatenation operator.
2. Secure and authentic transmission of the message M over the classical channel, with symmetric-key cryptographic primitives: M is encrypted with encryption key $K_{encrypt}$ and authenticated with the authentication key K_{auth}.

As previously mentioned, the only encryption scheme with provable information-theoretic security is the one-time-pad encryption, making it a logical choice to integrate with Quantum Key Distribution (QKD). Establishing an unconditionally secure classical communication link stands out as a vital application of QKD. Exploiting the constant secrecy provided by the one-time-pad and the unconditional security of QKD-generated keys, message encryption reaches a level of security that cannot be achieved without QKD in the key agreement process. The messages reach complete confidentiality against adversaries, ensuring robust resistance to any future events that might compromise their security.

Numerous specific scenarios demand long-term security [8], such as safeguarding medical records, industrial secrets, and classified military or governmental information. Present computationally secure schemes, however, cannot assure long-term security for highly sensitive data. Notably, when dealing with encrypted information transmission, adversaries can store ciphertext and await decryption until more advanced cryptanalysis methods or hardware emerge. For instance, the advent of more efficient factoring algorithms or breakthroughs in attacking Advanced Encryption Standard (AES) could pose risks. In this context, the fusion of QKD with the one-time-pad emerges as a practical solution, ensuring unconditionally secure data transmission over a point-to-point link. This integration serves as a natural and effective response to meet the strict requirements of high-security communication infrastructures, particularly in the context of long-term security.

3.2 QKD Networks

Implementations of Quantum Key Distribution (QKD) networks commonly depend on either optical switching or trusted relays. Alternatively, untrusted relays or quantum repeater, based solutions are also utilized. Among these, the optical switching and trusted relay schemes demonstrate a higher level of maturity compared to the untrusted relay and quantum repeater-based approaches [9].

3.2.1 Optical Switching Based QKD Networks

In a QKD network based on optical switching, classical optical functions like beam splitting and switching are applied to the quantum signals transmitted over a quantum channel, connecting a pair of QKD nodes. These functions can be readily implemented using commercial technologies. Quantum signals can traverse short quantum links without interacting with untrusted nodes, making these short links less susceptible to eavesdropping compared to long-range one. However, their applicability is limited to small-scale access networks and relatively compact metropolitan networks due to the inability to eliminate quantum signal attenuation through amplification [10] [11].

3.2.2 Trusted Relay Based QKD Networks

In contrast to the earlier described short-range setting, a QKD network based on trusted relays, commonly known as a trusted-node QKD network, involves generating local secret keys for each QKD link. These keys are then stored in nodes positioned at both ends of the respective QKD link. Enabling long-distance QKD between two end nodes, this setup relies on a one-dimensional chain of trusted relays connected by QKD links. The secret keys are transmitted from the source node to the destination node through a sequence of trusted relays in a hop-by-hop manner along the QKD path. The one-time pad technique is utilized for encryption, ensuring end-to-end information-theoretic security for the secret keys, as previously described. This implementation option for QKD networks is both practical and highly scalable, leading to widespread adoption for network deployment. It's essential to note that each trusted relay is assumed to be safeguarded against intrusion or any form of attack.

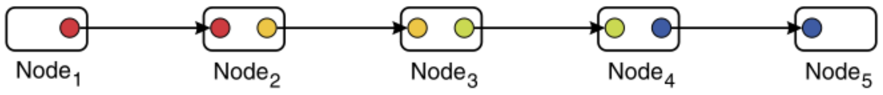

Figure 3.2: Hop-by-hop unconditionally secure message passing on a path made of trusted relay nodes connected by QKD links. Figure taken from [5].

Utilizing classical trusted repeaters allows the construction of a long-distance QKD network, and practical-scale implementations of such networks have already been showcased, initially with the DARPA Quantum network [12].

Figure 3.3 illustrates the four primary aspects of key relay. In Figure 3.3-A, it is evident that a key relay network runs parallel to a comprehensive network that carries communication messages and control traffic, including QKD protocols. In this context, the Internet serves as the communication network, with each link below representing a distinct QKD link, and circular nodes depicting key relay stations. In Figure 3.3-B, a specific source QKD endpoint (S) aims to establish key material with a distant destination QKD endpoint (D). As both endpoints, S and D, are linked to a ubiquitous communication network, they can engage in QKD protocols to derive key material. Once the keys are agreed upon, the Internet becomes the secure means of communication between them [12].

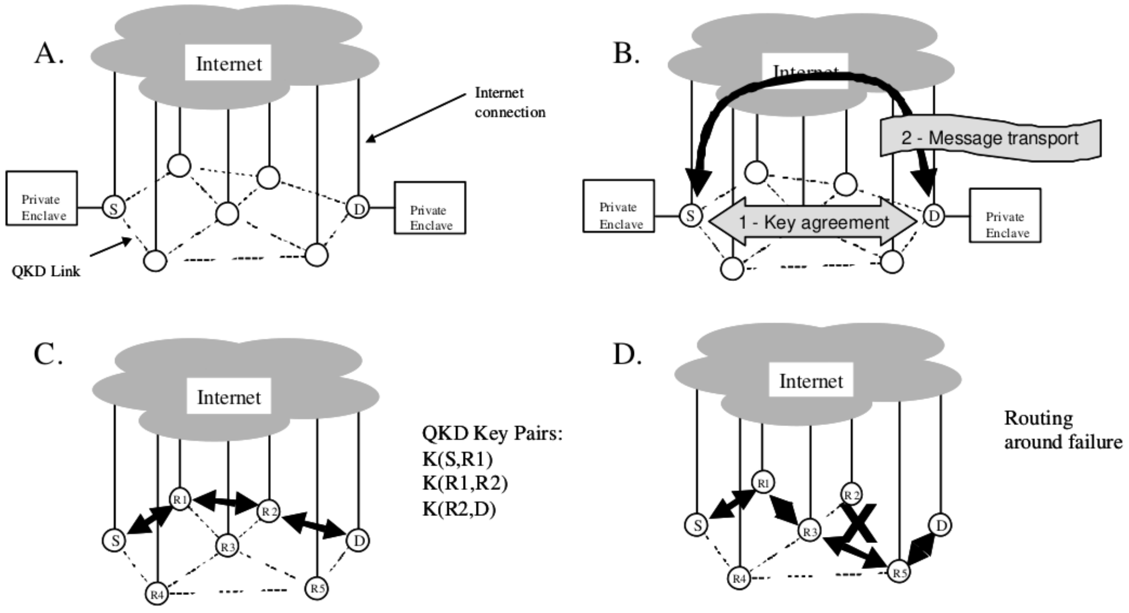

Figure 3.3: Major Aspects of the Key Relay, Figure taken from [12].

Figure 3.3-C illustrates the key relay path from S to D, with darkened lines representing the key relay network and resulting pairs of QKD key material on the right. Each QKD-derived key, such as $K(S, R1)$ between S and $R1$ or $K(R1, R2)$ between relay nodes $R1$ and $R2$, is denoted accordingly. With all pairwise keys established, S and D can easily derive their end-to-end shared secret key through key relay. One approach is for node S to generate a new random number R, protect it with $K(S, R1)$, and transmit the result to $R1$. $R1$ decrypts the message to obtain R, re-encrypts it with $K(R1, R2)$, and sends it to $R2$, repeating the process until reaching D. At this point, both S and D share the same secret random sequence, R, using it as key material [12]. The extension developed in the thesis uses a symmetric key for each pair of nodes which will be used to encrypt and decrypt the traffic, unlike the protocol just seen which uses this mechanism to bring a common secret key from node A to node B.

Lastly, Figure 3.3-D demonstrates that BBN key relay protocols [12] can autonomously detect failures along the key relay path, whether due to cut fiber or eavesdropping, and reroute the key material around these failures.

3.2.3 Untrusted Relay Based QKD Networks

A QKD network relying on untrusted relays necessitates the use of more secure QKD protocols, particularly those within the family of entanglement-based protocols. An untrusted relay based protocol is also capable of extending the secure distance of QKD quite considerably, this is because relays can be used as quantum repeaters [13][14]. Until now, a practical implementation of repeaters is not yet present.

3.2.4 Quantum Repeater Based QKD Networks

Quantum repeaters are used to tackle the issues caused by distance-dependent impairments affecting quantum signals [9]. Positioned at intermediate nodes, a quantum repeater utilizes a physical process called entanglement swapping to establish long-distance entanglement between source and destination nodes. The primary function of a quantum repeater is to purify and transmit quantum signals without directly measuring or cloning them. Despite the conceptual ideal of such quantum repeaters, as of the present writing [9], a practical implementation remains unavailable. Consequently, the deployment of long-range quantum repeater-based QKD networks in real-world scenarios is still pending.

3.3 Protocols

QKD protocols exhibit variations based on different criteria, with one of the most crucial factors being the method of encoding information for transmission. These protocols can be categorized into two main types: prepare-and-measure protocols and entanglement-based protocols.

Entanglement-based protocols utilize entanglement source Einstein-Podolsky-Rosen (EPR) particle pairs, while prepare-and-measure protocols, involve the preparation and measurement of quantum states for the secure transmission of information. For example, this type of protocol uses the polarization of photons to encode information inside, which will subsequently be measured to obtain the data. Furthermore these protocols perform error correction and privacy amplification after measuring the received quantum particle. This sequencing allows the extraction of classical information (a bit) initially. In the final step of QKD we perform the classical error correction. In this chapter, we will only explore protocols not relying on entanglement, specifically the Bennett-Brassard1984 (BB84) Protocol for QKD and Cascade protocol for error correction.

In this section we are going to see a high-level overview of the most well-known QKD protocols.

3.3.1 BB84 Protocol

The most widely recognized Quantum Key Distribution (QKD) scheme is the Bennett-Brassard1984 (BB84) protocol, as introduced by Bennett and Brassard in 1984 [7]. This protocol enables two users, namely Alice and Bob, who share a quantum channel (such as an optical fiber or free space) along with an authenticated classical channel, to establish a secure key even in the presence of an eavesdropper possessing unlimited quantum computing capabilities. In the BB84 protocol, Alice transmits a sequence of single photons, each carrying qubit states, to Bob through the quantum channel. Figure 3.4 presents a schematic diagram of the BB84 protocol [15].

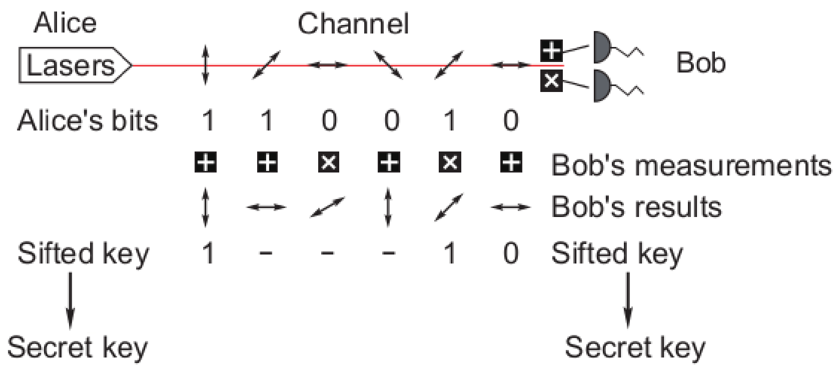

Figure 3.4: BB84 scheme. Figure taken from [15].

The steps of the protocol are the following:

1. Alice randomly encodes a single photon with one of four polarization states-vertical, horizontal, 45-degree, or 135-degree-for each signal. She then sends the photon through a quantum channel to Bob.
2. For each signal, Bob selects one of two bases-rectilinear and diagonal-to conduct a measurement on the polarization of the received photon. After detection, Alice and Bob publicly disclose their basis choices through an authenticated classical channel.
3. The polarization data encoded and detected in different bases are discarded by Alice and Bob. Only the data in the same basis are retained to form the sifted key. From this sifted key, Alice and Bob randomly select a sample to compute the Quantum Bit Error Rate (QBER).
4. If the computed QBER surpasses a certain threshold, they abort the process. Otherwise, they proceed with classical post-processing, including error correction and privacy amplification, to generate a secret key.

3.3.2 B92 Protocol

In 1992, Charles Bennett introduced the B92 protocol [16]. The B92 protocol represents a modified iteration of the BB84 protocol, distinguishing itself primarily in the choice of polarization states.

3.3.3 Information Reconciliation Protocol

The classic protocols seen previously consist of both a quantum phase and a classical post-processing phase. The quantum phase involves the utilization of both the quantum channel and the classical channel for key exchange. The subsequent classical post-processing phase exclusively relies on the classical channel and consists of two distinct components [12]:

1. Information reconciliation, tasked with identifying and rectifying unavoidable bit errors (noise) in the key exchanged during the quantum phase.
2. Privacy enhancement, responsible for minimizing information leakage during the information reconciliation step.

This section focuses solely on a particular information reconciliation protocol, namely the Cascade protocol.

Key bit errors

Key distribution protocols invariably introduce noise into the key. The key received by Bob exhibits some noise, such as bit errors, in comparison to the key originally sent by Alice. Consequently, we label the key transmitted by Alice as the correct key, and the key received by Bob as the noisy key.

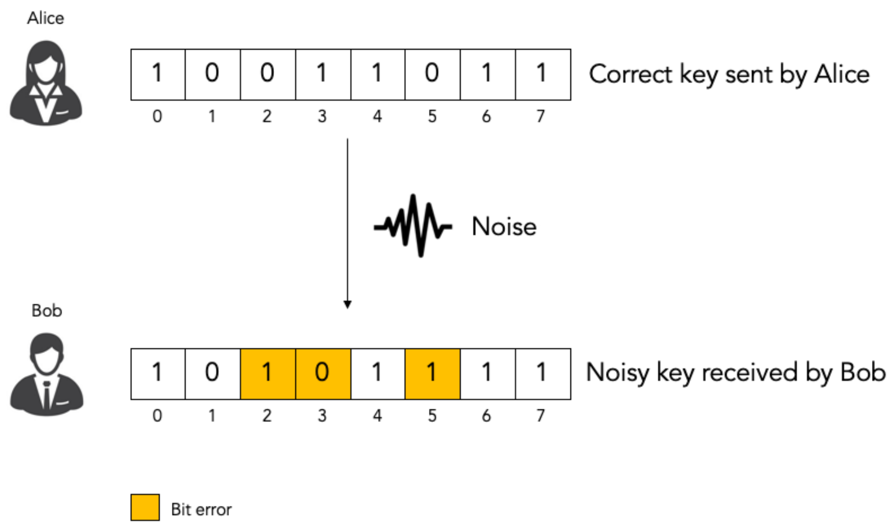

Figure 3.5: Example of noisy key, Figure taken from [20].

Noise may arise from imperfections in hardware or random environmental fluctuations. Alternatively, eavesdropper Eve observing traffic can also introduce noise. It's crucial to note that in quantum mechanics, the act of observing a photon induces changes, leading to detectable noise. Quantum key distribution protocols universally provide an assessment of the noise level, typically expressed as an estimated bit error rate. A bit error rate of 0.0 indicates that no key bits have been altered, while a rate of 1.0 signifies that all key bits have undergone changes.

Classical post-processing

When the estimated bit error rate surpasses a predefined threshold, it leads us to the inference that Eve is monitoring the traffic, attempting to decipher the secret key. Under such circumstances, we opt to discontinue the key distribution attempt. Conversely, if the estimated bit error rate falls below the threshold, we proceed with classical post-processing. This involves the two steps mentioned earlier, Information reconciliation and Privacy enhancement, both of which are classical protocols, exclusively involving classical communications and void of any quantum communications.

Information reconciliation

The initial step in classical post-processing is information reconciliation [12]. Even when the bit error rate falls below the threshold, it is not reduced to zero.; there remains some noise in the form of bit errors within the noisy key received by Bob compared to the correct key from Alice. The objective of the information reconciliation step is to identify and rectify these residual bit errors.

There are different ways to reconcile information, and here, we're focusing on one called the Cascade protocol. The challenge in this process is to prevent too much information from being leaked. If someone, like Eve, who's trying to eavesdrop, gets hold of any leaked information during reconciliation, it could make it easier for them to decrypt the encrypted messages. Even if Eve doesn't get the entire key, having some leaked information about it makes it simpler for her to try fewer options when attempting to break the code. Each piece of leaked key information significantly reduces the number of attempts Eve needs to make during a brute force attack [12].

Nevertheless, it's inevitable that the information reconciliation protocol will leak a limited amount of information. This is acceptable as long as the leaked information is both bounded and known, allowing us to compensate for it.

The Cascade protocol

The Cascade protocol serves as an illustration of an information reconciliation protocol. Its primary objective is to identify and correct any persistent bit errors within the noisy key received by Bob in comparison to the correct key sent by Alice.

Consider a scenario where Alice and Bob, having completed the quantum phase of a quantum key distribution like BB84, find themselves in possession of the correct key and a noisy key, respectively. The noisy key, although resembling Alice's correct key, contains a limited number of bit errors. Subsequently, Alice and Bob initiate the Cascade protocol to pinpoint and correct any remaining bit errors within Bob's noisy key [12].

Cascade operates entirely within the classical channel, exclusively relying on the exchange of classical messages. It does not engage in any quantum communications. Traditional techniques, such as TCP/IP, are employed to ensure reliability, flow control, and other functionalities. It is not a prerequisite for the classical channel to be encrypted; we operate under the assumption that eavesdropper Eve **can openly observe** all classical messages.

Nevertheless, it is essential for the classical channel to offer authentication and integrity. We presume the existence of a mechanism enabling Alice and Bob to confirm that all classical messages originated from the legitimate sources, Bob and Alice, and have not been manipulated or forged by Eve. This requirement is crucial in preventing man-in-the-middle attacks by Eve, where she intercepts all classical traffic, impersonating Bob to Alice and vice versa.

Cascade operates through iterations involving random permutations to evenly distribute errors within the sifted key. The permuted sifted key is initially divided into equal blocks of size k_i bits. After each iteration and new permutations, the block size doubles to $k_i = 2{\cdot}k_{i-1}$. Parity tests are conducted for each block, and a binary search is employed to identify and rectify errors within the block. To enhance efficiency, the Cascade protocol explores errors in pairs of iterations through a recursive approach.

Rather than outright rejecting error bits in the initial stage, information regarding the existence of an error bit within the block is harnessed in subsequent iterations to identify errors overlooked due to measurement parity. Any errors discovered in later iterations enable the identification of at least one corresponding error in the same block from the previous iteration, initially presumed error-free. Through a binary search, an extensive exploration for errors in such a block is conducted, facilitating the recursive detection of masked errors. The illustration in Fig. 3.6 depicts two passes of the Cascade protocol.

The Cascade protocol depends on employing a binary search for pinpointing error bits. This binary search involves iteratively dividing the block into two smaller subblocks, where the parity check values are compared until the identification of an error.

The Figure 3.6 illustrates the first two steps of the Cascade protocol. In step 1, as seen in the upper part of the image, Alice has sent bits to Bob, and now both have a sequence of bits representing a key. Due to noise, some bits received by Bob are incorrect. The two streams of bits are divided into blocks of a length of $k_1 = 4$. Now, Bob locally computes the parity of each block, where the parity value is 1 if the number of '1's in the block is odd. Parity is used to reveal only a small amount of information about the key; hence, we cannot have an initial block size of 1, as it would leak the entire key. Once the parity bits are calculated, Bob compares them with the parity bits of Alice's blocks. If he finds a block with different parity bits, he knows there is at least one error. Bob further divides the block in half and again compares the parity bits of the two sub-blocks to identify the error until he obtains a single block and corrects the bit. At the end of this first step, we have corrected only one bit. If there are no other blocks with different parity bits, we proceed to the next step, as now all blocks have correct parity compared to Alice's blocks. In the second step shown in the figure, we observe that the bit sequence has been randomly reordered. This is necessary and helpful to recalculate the parity bits of the blocks, as two blocks may have the same parity bit but contain errors. Bob must inform Alice of the reordering, for example, with a list of indices of permuted bits. Additionally, in the second step, the block size is doubled $k_2 = k_1 \times 2 = 8$. The block size is doubled because in the previous step, some bits were corrected, resulting in a lower error rate. The same search process will be used in this step as well. At the end of this second step, if we are fortunate, the entire key has been corrected. The original implementation of Cascade uses 4 iterations, at the end of which it is highly likely that all errors have been corrected.

The cascade protocol also includes a so-called privacy amplification phase, for each parity bit sent on the classic channel for a block, a bit of the block will have to be discarded since the parity bit reveals information about the block.

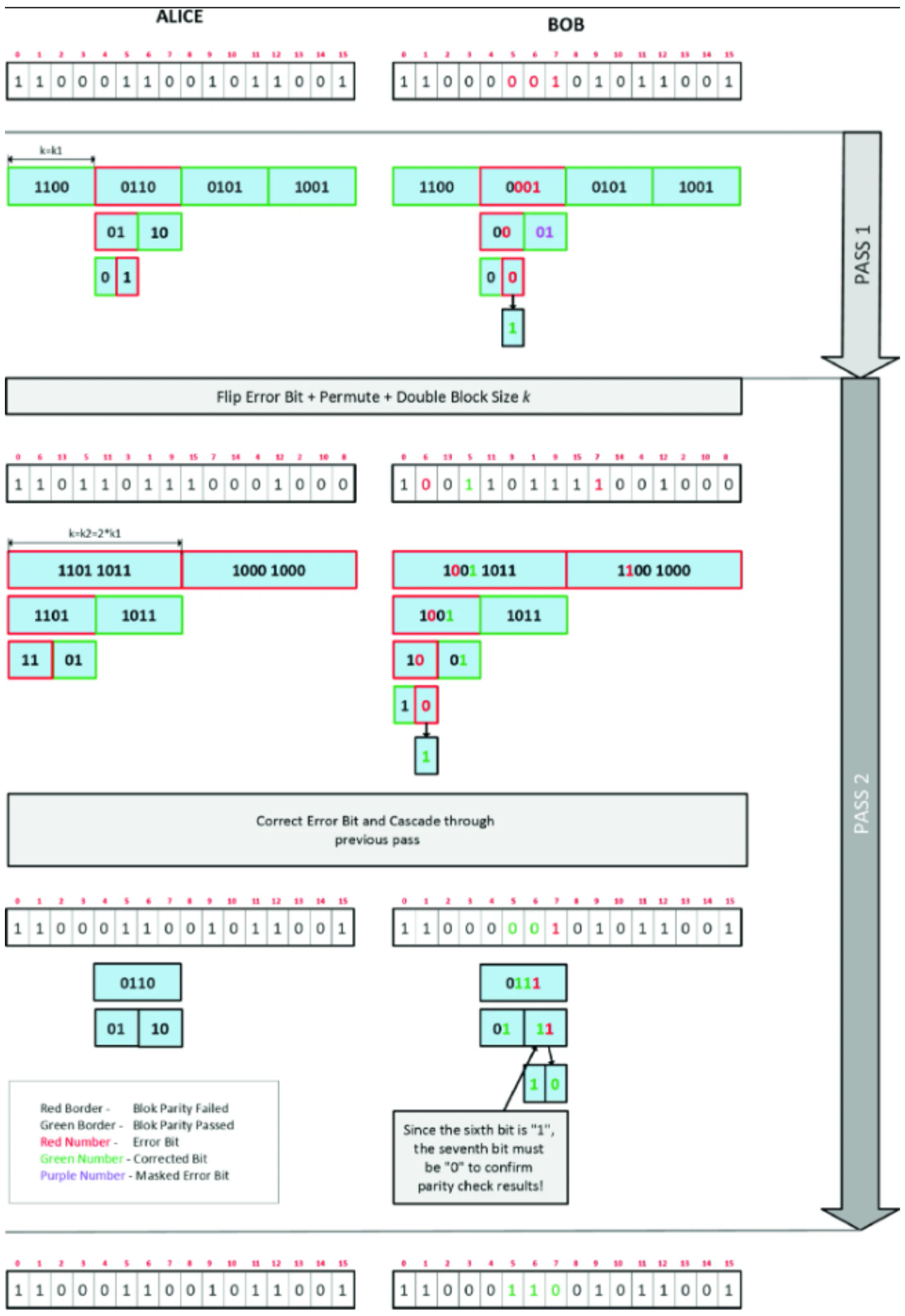
ALICE
BOB
k=k1
PASS 1
Flip Error Bit + Permute + Double Block Size *k*
k=k2=2*k1
PASS 2
Correct Error Bit and Cascade through previous pass
Red Border - Blok Parity Failed
Green Border - Blok Parity Passed
Red Number - Error Bit
Green Number - Corrected Bit
Purple Number - Masked Error Bit
Since the sixth bit is "1", the seventh bit must be "0" to confirm parity check results!

Figure 3.6: Illustration of the first two passes of reconciliation using a Cascade protocol. Figure taken from [21].

Chapter 4

QKD Simulator

In this chapter we will talk about the type of simulation used and the SeQUeNCe (Simulator of Quantum Network Communication) simulator [18]. Simulations provide a controlled environment to explore and understand the behavior of Quantum Key Distribution (QKD) systems before they are implemented in real environments. However, it is crucial to experimentally verify implementations in the laboratory to ensure the safety and effectiveness of the system in practice. QKD is an ever-evolving field, with numerous theoretical and experimental developments seeking to address challenges and improve the performance of the technology.

4.1 DES - Discrete Event Simulation

Lots of computer and communication systems use something called discrete event simulation. Basically, it involves using a global time called *currentTime* and an event scheduler. In simple terms, events are like actions that represent different changes, and each has a specific time when it happens. The event scheduler keeps a list of these events in order of when they're supposed to happen.

During simulation, the program selects the first event from the scheduler, advances *currentTime* to the firing time of this event, and executes the event. Event execution may result in scheduling new events with firing times greater than or equal to *currentTime*, as well as modifying or deleting events previously listed in the scheduler. It's important to note that the global simulation time, *currentTime*, remains unaltered by an event, causing the simulation time to transition discretely from one event firing time to the next [19].

4.2 SeQUeNCe: Simulator of QUantum Network Communication

SeQUeNCe (Simulator of Quantum Network Communication) is an open source quantum simulator written in Python developed by Argonne National Laboratory [18].

The SeQUeNCe simulator has the following features, as reported in the Paper [18]:

- **Realism of Quantum States**: The simulator should accurately trace quantum states, including entanglement, and assess their fidelity. Quantum states may be encoded using methods such as time bins, polarization of light, or states in quantum memories. The quality of entanglement, a crucial metric for quantum network performance, requires modeling for factors like loss and decoherence. In SeQUeNCe, entanglement states are represented as complex arrays, and the simulator includes hardware models to record entanglement fidelity.

- **Realism of Timing**: Simulation events must be executed precisely at their designated timestamps and in their exact order to prevent causality errors. Quantum networks are time-sensitive, and the arrival times of photons, which encode quantum information, determine their identity. Furthermore, the limited lifetime of qubits in memories necessitates low-latency operations. The simulator operates with picosecond precision to meet these temporal requirements.

- **Flexibility**: For the development of future quantum networks, the simulator needs to simulate diverse network architectures, new protocols, and applications, while allowing reconfigurable topologies and traffic traces. SeQUeNCe employs a modularized design, separating functionality into modules containing reprogrammable protocols. This design facilitates quick testing of numerous scenarios by adjusting parameters in JSON files.

- **Scalability**: The need arises for large-scale studies of wide area networks with numerous components and the tracking of quantum states at the individual photon level. In contrast to classical packet-level network simulations, the simulation involves tracking photons generated at megahertz frequencies, leading to a significant increase in the number of simulated events. Additionally, a stand-alone simulation kernel has been designed to enable portability to high-performance computing systems.

4.2.1 Modularized Design of SeQUeNCe

In simulating quantum networks, certain assumptions must be made about their architecture. However, it's noteworthy that quantum network architectures have not yet been standardized, and ongoing discussions on this matter are taking place within the recently formed Internet Engineering Task Force (IETF) standardization group [18]. SeQUeNCe is structured around a modular design, as in Figure 4.1, employing six modules to generate events. The following models are taken from the SeQUeNCe paper [18].

- **Simulation Kernel** acts as the core of SeQUeNCe, facilitating discrete-event simulation. Simulation time progresses in discrete clock ticks, and events generated by simulation models in other modules are stored in a priority queue, sorted by their timestamps. Continuously, the kernel executes the top event in the queue, advancing the simulation time accordingly. This process repeats until the queue is empty or a specified simulation end condition is reached. The Kernel offers users significant control over event execution orders for realistic timing and includes interfaces for potential future parallelized implementation to enhance scalability.

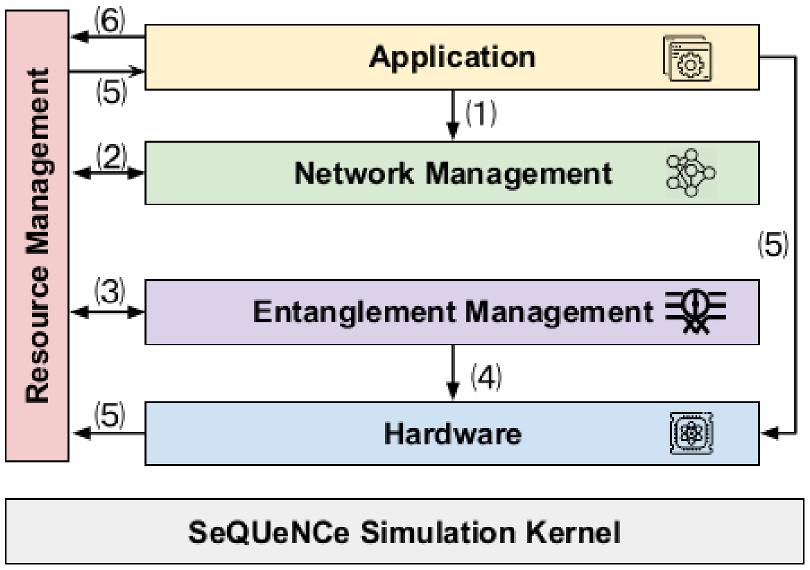

Figure 4.1: Modularized design of SeQUeNCe, figure taken from [18].

- **Hardware** module includes models of elementary hard- ware components used in a quantum network, including quantum channels, classical channels, quantum gates, photon detectors, and quantum memories.
- **Entanglement Management** module includes models of protocols for reliable high-fidelity end-to-end distribution of entangled qubit pairs between network nodes.
- **Resource Management** module manages local resources within one node. It records the state of the hardware, efficiently allocates resources to applications and entanglement protocols based on commands issued by the network management, and regains control of the hardware with updated states when resources are released.
- **Network Management** module provides quantum network services based on requests from the local Applications and remote Network Managers.
- **Application** module represents quantum network applications and their requests for quantum network resources.

Chapter 5

Implementation

This chapter delves into the development process of the SeQUeNCe simulator extension, specifically designed for analyzing the performance of Quantum Key Distribution (QKD) using Free-Space Optical (FSO) Links. Subsequent chapters provide a comprehensive exploration of node structures, links, messaging protocols, and routing details. Commencing with an overview of sequence graph structures, we explore how these graphs are customized to suit simulation requirements with the extension.

5.1 Introduction

SeQUeNCeuses, as anticipated, a customized structure to represent a network. For the purposes of our extension, we would be interested in simulating graphs with specific characteristics (for example, a certain number of nodes, distance between nodes, channel bit-rate, BER, etc.). NetworkX is a powerful and popular Python package designed for the creation, manipulation, and analysis of complex networks and graphs. It provides a comprehensive set of tools and functions that enable users to study various aspects of network science, social network analysis, and graph theory [22]. NetworkX allows us to generate custom graphs by specifying some characteristics. However these graphs are not compatible with SeQUeNCe. In fact, to allow us to use these types of graphs in our extension, a simple parser has been created that "translates" a NetworkX graph into a SeQUeNCe-compatible graph.

At its core, NetworkX represents networks as collections of nodes (also known as vertices) and edges (also known as links or connections) that connect these nodes. NetworkX supports a wide range of network types, including directed and undirected graphs, multi-graphs (graphs with multiple edges between the same nodes), and graph sequences. Moreover NetworkX provides convenient functionality to export networks and graphs to JSON files, allowing users to store and share their network data in a widely supported format. With NetworkX, you can export a network to a JSON file using the json_graph module, which is part of the NetworkX package. This module provides functions to convert NetworkX graphs into JSON-compatible data structures.

To export a network to a JSON file, is possible to use the `networkx.json_graph.node_link_data` function. This function takes a NetworkX graph as input and returns a dictionary representation of the graph that can be easily serialized to JSON.

So to ensure compatibility between NetworkX's output and SeQUeNCe, a dedicated parsing function was created, for implementation refer to the repository. This converts the JSON output from NetworkX into a modified JSON format that SeQUeNCe can understand. The modified JSON file retains the original network topology while being formatted in a way that allows SeQUeNCe to load the network topology along with its internal objects. By utilizing this parser, we ensure a base integration between the generated network topology and the functionality provided by SeQUeNCe.

We can see in listings 5.1 and 5.2 an example of a graph in NetworkX and one in sequence obtained from the parsing functions. The graph will be populated with predefined attributes, for more specific simulations the attribute of interest will have to be modified. The various attributes will be explained in the following sections.

```
{
    ...
    "nodes": [
        {
            ...
            "id": 0
        },
        ...
    ],
    "links": [
        {
            ...
            "source": 0,
            "target": 1
        },
        ...
    ]
}
```

Listing 5.1: NetworkX Graph Example

```
{
    "nodes": [
        {
            "name": "node0",
            "type": "QKDNode"
        },
       ...
    ],
    "qchannels": [
        {
            "name": "qchannel0_0to1",
            "source": "node0",
            "destination": "node1",
            "distance": 1000,
            "bit_rate" : 1000000000
        },
        ...
    ],
    "cchannels": [
        {
            "name": "cchannel0_0to1",
            "source": "node0",
            "destination": "node1",
            "distance": 1000,
            "bit_rate" : 1000000000
        },
        ...
    ]
}
```

Listing 5.2: Sequence Extension Graph Example

5.2 QKD Link

In this section we will see the details of the implementation of the quantum and classical channels, with more attention on the quantum channel. The class that deals with modeling optical channels is OpticalChannel and is implemented by the SeQUeNCe simulator. For our objectives we are interested in modeling an FSO Link, so for the implementation we used the class offered by SeQUeNCe as support, but we also customized some parameters.

Let's see an overview of how an optical channel is made up in SeQUeNCe and then let's see how it is used in the extension.

5.2.1 ClassicalChannel

The classic channel, used for classic message exchange, is implemented very simply through the following parameters, for more details refer to the SeQUeNCe documentation:

- **Sender**: node at sending end of optical channel.
- **Receiver**: node at receiving end of optical channel.
- **Distance**: length of the fiber (in m).
- **Delay**: delay (in ps) of message transmission.

The channel is very simple, the delay is calculated based on the distance and speed of light in the fiber, so that it is possible to schedule a transmission delay in the simulation. The transmission is carried out by scheduling a message event received on the recipient node.

5.2.2 QuantumChannel

The quantum channel is more complex, some of the parameters that characterize it are the following:

- **Attenuation**: attenuation of the fiber (in dB/m).
- **Polarization Fidelity**: probability of no polarization error for a transmitted qubit.
- **Frequency**: maximum frequency of qubit transmission (in Hz).

The quantum channel is used exclusively by QKD protocols. For the BB84 for example it is used to send polarized photons. A very relevant parameter for the quantum channel is the *Polarization Fidelity*, which specifies the probability that there are no polarization errors of the photon at the moment of sending. Being a probability, its value will be within an interval ranging from 0 to 1. So for example, once the photon has been prepared by encoding the information to be transmitted, if the Polarization Fidelity is set to 1, then when the photon is sent there will be no noise applied to the photon. While if we have a Polarization Fidelity of 0.5 then we expect that the probability that noise will be applied to the photon is 50%.

5.2.3 Free-Space Optic (FSO) Link

SeQUeNCe does not provide a class that models Free-Space Optic (FSO) Link, so to model one of these links a QuantumChannel appropriately configured with adequate parameters was used. Let's see briefly what an FSO Link is.

An Free-Space Optic (FSO) Link, which stands for Free-Space Optical Link, is an optical communication system that transmits data through free space, typically using laser light beams. This technology is also known as free-space optical communication or optical communication through the air. The operation of an FSO Link involves sending optical signals through the atmosphere without the use of cables or optical fibers. FSO devices consist of laser transmitters and optical receivers positioned at two communication points. The light signals travel through free space and are received by the other device, where they are then converted back into electrical signals for data processing.

This type of technology can be used for short-distance communication between buildings, university campuses, or other limited geographical areas. However, free-space optical communication can be influenced by environmental factors such as fog, rain, and other weather conditions that may attenuate or disrupt the optical signal. FSO Links are often chosen when the installation of physical cables (such as optical fibers) is impractical or too costly. They are also used in scenarios where a high-speed connection is required, and the distance between communication points is relatively short.

To model the FSO Link, we are going to use an instance of a QuantumChannel. What we are interested in knowing about our link will be the distance, a BER (Bit error rate) and the frequency. With a model built for FSO Link, within the FSOQKD class, we estimate a BER through the distance and frequency (bit rate) of the link. To model the BER of the FSO link we are going to use the BER value obtained from the model mentioned above. In fact, this will represent a fundamental value for the link, as it will influence the percentage in which the bits are altered. To do this, the source code was modified when generating the bits to be sent before the BB84 protocol, as in Listing 5.3.

```
bit_list_noise = []

for i in range(num_pulses):
    if numpy.random.random(1)[0] < self.ber:
        bit_list_noise.append(1 - bit_list[i])
    else:
        bit_list_noise.append(bit_list[i])
```

Listing 5.3: Code to Model BER in SeQUeNCe

5.3 Design of a QKD node

In this section, we will explore the idea behind how nodes are modeled within the simulation.

SeQUeNCe provides us with QKDNodes that have all the hardware setup necessary for the execution of a QKD protocol, making our lives much easier. However, we encountered a problem with this setup. Suppose you have three nodes named *node0*, *node1*, and *node2*, and you want each of them paired with each other to exchange quantum keys. This proves challenging due to the strict sender-receiver roles each QKDNode assumes in its protocol stack. In practical terms, if *node0* is paired with *node1* where *node0* is the sender and *node1* is the receiver, then we cannot have *node0* paired as the sender with node3 or any other node. This restriction arises because its internal protocol stack encounters simulation conflicts in such scenarios.

For this reason some wrappers have been created around the SeQUeNCe QKDNode. The first wrapper is the Transceiver class, which is nothing more than a wrappers for the QKDNode, which will be used to connect to a Transceiver on another adjacent one. The second wrapper is the class SuperQKDNode, this structure will contain a Transceiver for every other node it is adjacent to. The Figure 5.2 will clarify these structures, with an example of two connected SuperQKDNodes. An example SeQUeNCe graph of this simple network is shown in Listing 5.4.

```

    "nodes": [
        {
            "name": "node0",
            "type": "QKDNode"
        },
        {
```

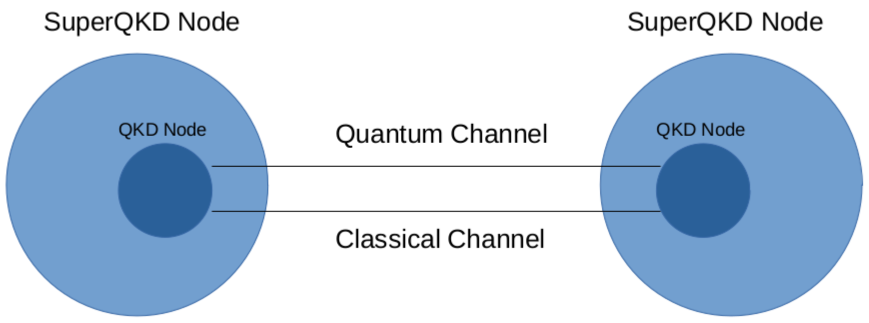

Figure 5.1: Abstract representation of a super QKD node.

```
                "name": "node1",
                "type": "QKDNode"
            }
        ],
        "qchannels": [
            {
                "name": "qchannel0_0to1",
                "source": "node0",
                "destination": "node1",
                "distance": 1000,
                "bit_rate" : 1000000
            },
            {
                "name": "qchannel0_1to0",
                "source": "node1",
                "destination": "node0",
                "distance": 1000,
                "bit_rate" : 1000000
            }
        ],
        "cchannels": [
            {
                "name": "cchannel0_0to1",
                "source": "node0",
                "destination": "node1",
                "distance": 1000,
                "bit_rate" : 1000000
            },
            {
                "name": "cchannel0_1to0",
                "source": "node1",
                "destination": "node0",
                "distance": 1000,
                "bit_rate" : 1000000
            }
        ]
}
```

Listing 5.4: SeQUeNCe-Extension 2 node graph example

The Transceiver class, as previously mentioned, is a wrapper around the QKDNode class, in fact an instance of this structure will contain a QKDNode, a reference to the messaging protocol and one to the key Manger. While a SuperQKDNode instance will contain a collection of Transceivers and a routing table, for routing the various messages between the SuperQKDNodes.

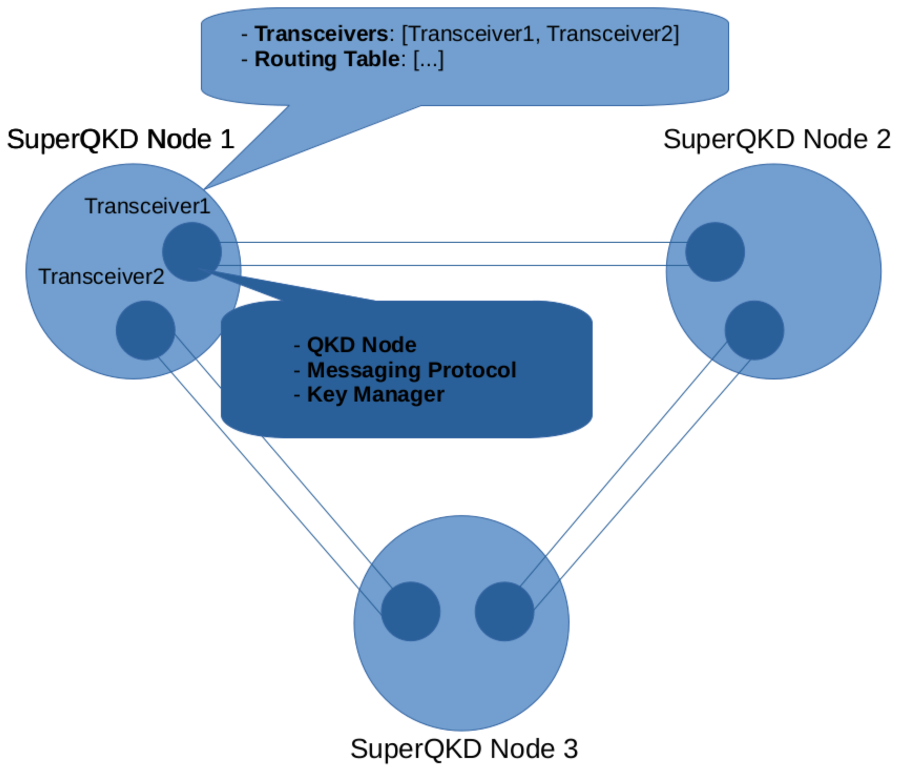

Figure 5.2: Abstract representation of a SuperQKDNode and Transceivers.

Therefore the QKD will take place between the pairs of Transceivers (therefore the exchange of keys through the BB84 and Cascade protocols) and the exchange of messages. So the SuperQKD Node will be responsible for routing and will delegate message forwarding to the Transceivers, this will be explored in more detail when we talk about routing. As discussed previously, each Transceiver has a Key Manager, who will be responsible for starting the protocols for key agreement and key storage.

The key manager is modeled in the following way, it has three methods (for more detail refer to the repository):

1. **send_request**: which takes care of starting the QKD between two paired QKD nodes.
2. **pop**: which is an interface for Cascade (class implemented by the SeQUeNCe simulator) to return the generated keys.

3. **consume**: which takes care of consuming a key, of the generated and saved keys when sending a message.

Each QKD Node, within a Transceiver, has a LightSource, the LightSource component acts as a simple low intensity laser. This component is responsible for emitting photons towards the recipient, as can be seen in the Figure 5.3, as seen in the previous chapter, the extension uses a QuantumChannel with peronslaizated parameters to model an FSO link. LightSource is configured with the following parameters:

- **Frequency**: frequency (in Hz) of photon creation.
- **Mean photon number**: mean number of photons emitted each period.

The light source frequency is the one defined in the FSO Link (QuantumChannel). A period is defined by the frequency of the component expressed in picoseconds, i.e. $1e12/Frequency$.

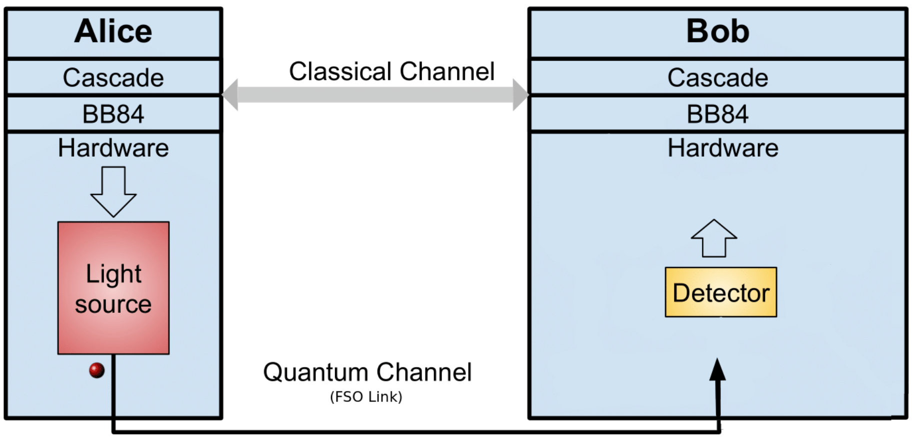

Figure 5.3: High-level view of the sending and receiving components, figure taken and modified from [17].

5.4 Messaging Protocol

In this section we will see how the Messaging Protocol was developed. Each Transceivers, as shown in Figure 5.2, has an instance of MessagingProtocol which is the class responsible for sending and receiving messages through the classic channel. This class is a subclass of the Protocol class of SeQUeNCe, this class defines the protocol type inherited by all implementations of the protocol code.

Through the `start_messaging` function of the Topology class of the extension it is possible to start the exchange of messages between the nodes of the network. This function will generate random traffic, establishing a destination for each node. It is also possible to establish customized traffic through a json file that defines a destination for each node, as shown in Listing 5.7.

```

{
    "packets" : [
        {
            "src_node" : "node0",
            "dst_node" : "node1"
        }
    ]
}
```

Listing 5.5: Example of custom traffic of a network of 2 nodes

The sending of messages will be determined by a rate that defines the load, i.e. how many packets each node generates for sending in one second. This rate is the average of an exponential random variable. Furthermore, the sending of messages is also limited by the capacity of the classic channel, i.e. its frequency or bit rate. We see that some fundamental parameters of the protocol are the following, for more details refer to repository:

- **mess_rate**: it is the rate that defines the load, the average of an exponential random variable.
- **packet_period**: defines the period for transmitting a packet. It is calculated starting from the bit rate of the classic channel and the size of a packet.
- **buffer_capacity**: queue capacity.

Sending a packet depends on the presence of keys to encrypt it and the availability of the queue. Once a packet has been generated, it is checked whether there is space in the queue, if there is space, the packet is queued, otherwise it is dropped. Sending will only be possible if there are keys. If a key is present, then a packet is extracted from the queue which is encrypted with it and sent to the destination. In the simulation it is assumed that the packet is the same size as a key, so if we have 128 bit keys we will have that the packet will be 128 bits. A packet therefore consumes a key.

In addition to a packet generated by the node itself, it is possible that a packet may also arrive from another node and need to be forwarded, if the destination is not the node itself. Then node checks whether there is space in the queue, then it can be added to the queue or dropped. We discuss Routing in the next section.

5.5 Routing

In this section, we will explore how routing is implemented. Once the network is generated, a path is calculated for each SuperQKDNode to reach every other node in the network, using the NetworkX `shortest_path` function. This way, each SuperQKDNode will know the path to reach every other node in the network. As mentioned in previous chapters, it is the responsibility of the SuperQKDNode to route packets, which involves selecting the correct Transceiver for sending or forwarding packets.

Through the method `send_message(self, tl, dest_node, plaintext_msg, forwarding)` in the SuperQKDNode class, it's possible to send a message to the destination node. The method takes parameters such as the name of the destination node, the packet, and a forwarding parameter. The forwarding parameter is necessary for the extension to understand whether it's a forwarded message or a message generated by the node for sending. In Listing 5.6, we can see the format of a packet. The fields *hop* and *time* are two fields that will be modified. The *time* field will be set to the sending time when the packet is sent, and the *hop* field will be incremented with each hop the packet makes.

```
{
    "src": "node0",
    "dest": "node1",
    "payload": "text",
    "hop": 0,
    "time": None
}
```

Listing 5.6: Example of Packet format

So, once the `send_message` method is called, the SuperQKDNode will check its routing table and delegate the sending or forwarding of the packet to the Transceiver connected to the next hop. Once the receiving Transceiver gets the packet, the messaging protocol's job is to check if the packet has reached its destination or if it needs to be forwarded. If the packet needs to be forwarded, then the messaging protocol will delegate forwarding to the SuperQKDNode, calling the `send_message` method with the forwarding parameter set.

When a packet needs to be sent, whether it's forwarding or sending a packet generated by the node, it gets placed in a queue. The packet transmission depends on the presence of keys. So, all packets will be queued in the respective Transceiver's queue and sent using the keys generated by the QKD for that specific Transceiver and its destination. If the queue is full, incoming or generated packets will be dropped. Figure 5.4 illustrates how sending messages works after the SuperQKDNode picks the right Transceiver for sending, and it also shows how message reception behaves.

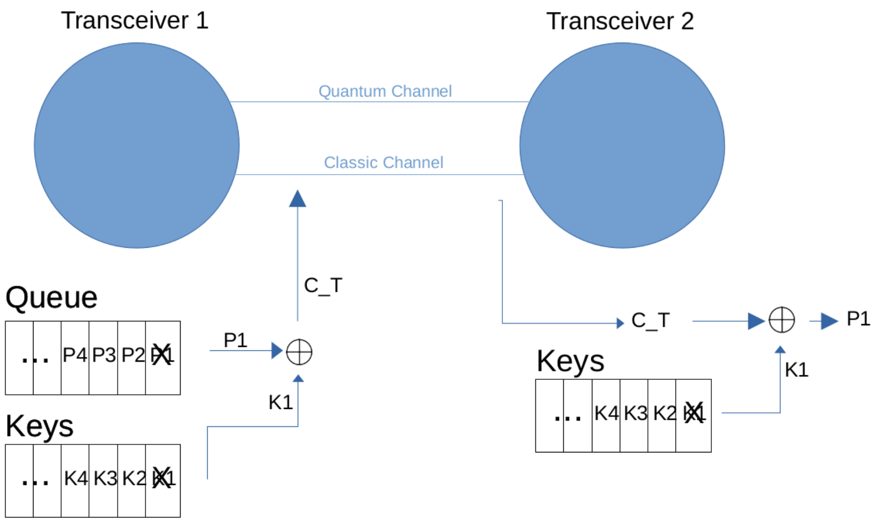

Figure 5.4: Sending and receiving a packet.

5.6 Simulations

In this section, let's see how to set up a simulation and the various outputs that it will return. Each run will generate a folder in the *sim directory* folder, named with the current timestamp, in which are saved files containing the simulation results. Some outputs depend on the types of input parameters. For example, if a graph of a Sequence network is provided directly, the simulator will not return a graphical image of the network, as this is possible if the graph is first generated with NetworkX.

Simulation parameters are customizable using the following CLI arguments when launching a simulation:

- **--sim-time SIM_TIME**: Establishes the simulation time in seconds.
- **--netx-graph NETX_GRAPH**: Specifies the path to a network json file in NetworkX.
- **--seq-graph SEQ_GRAPH**: Specifies the path to the json file of a network in Sequence.
- **--key-size KEY_SIZE**: Indicates the length of keys that QKD will generate for node pairs.
- **--mess-rate MESS_RATE**: Specifies the load rate, i.e. how many packets each node will generate per second on average, times generated by an exponential random variable with mess-rate as parameter.
- **--num-nodes NUM_NODES**: If we want to generate a random network with NetworkX, this parameter indicates how many nodes we want in our network.

- **--buff-capacity BUFF_CAPACITY**: Indicates the capacity of the buffers in the nodes.
- **--inspection-rate INSPECTION_RATE**: This rate is used to inspect the buffers and keys in each node, that is, how often we visit the nodes to obtain information.
- **--traffic TRAFFIC**: A path must be specified to a json file that contains custom traffic.

In addition to the parameters for customizing a simulation, customization also occurs through the json files that represent the network. In fact, as seen in Listing 5.4 , with these files it is possible to establish the bit rates of the classical and quantum channels and the lengths, through which the BER of the quantum channel will subsequently be calculated.

Now to give an example of use, let's say that we want to simulate on the Listing 5.4 network, for a second of time, the exchange of 128 bit messages and keys with a *mess_rate* of 10 packets per second and personalized traffic as in Listing 5.7. Then the command line will be:

```
python3 project/sim_ext.py --seq-graph sequence_2_node_graph.json --traffic
    traffic.json --inspection-rate 0.001 --sim-time 1.0 --mess-rate 0.1
```

Listing 5.7: Example of simulation command

The structure of the simulation folder is as in Figure 5.5, in which are saved the JSON files containing the network topology, a JSON that contains the simulation parameter, the image containing the graphical representation of the network, one csv file that contains the capacity of the buffers of the nodes during the simulation, a csv that contains information on the packets and a csv that contains for each node the total number of keys exchanged during the simulation. Let's take a look at the format of the files resulting from the simulations:

- *packet_result.csv*: The file contains all the information about the generated packets, indicating whether they were sent, delivered, or dropped. Additionally, it includes the source and destination nodes, the number of hops, and the send and receive times. Listing 5.8 shows the header of the csv file.

  ```
  Sim. Command,Source,Destination,Sent,Delivered,Dropped,Num. Hop,Sending
      Time,Sim. Time,Tot. Time
  ```

  Listing 5.8: Headers of packet_result.csv file

  The headers Source and Destination will contains the name of the nodes. The headers Sent, Delivered and Dropped are boolean headers, so for example if the packet is sent the header will be True. If the packet is dropped the Dropped header will contains the name of the Transceiver that dropped it.

- *buffers.csv*: Based on the inspection time parameter, in this file, we will find, for each Transceiver, the buffer length, meaning how many packets are in the buffer at that moment. The headers of this file will be the names of all the transceivers, and in the rows, you'll find the corresponding buffer length for each transceiver.

- *tot_keys.csv*: In this file, we will find, for each Transceiver, the total number of keys generated. The headers of this file will be the names of all the transceivers, and in the rows, you'll find the corresponding total keys generated for each transceiver.
- *sim_params.json*: In this json file we will find a resume of the values of the parameter. Listing 5.9 shows an example of this json file.

```
{
    "sim_time": 1.5,
    "netx_graph": null,
    "seq_graph": "project/file/graph_15_nodes.json",
    "key_size": 128,
    "mess_rate": 0.002,
    "num_nodes": 10,
    "buff_capacity": 10,
    "inspection_rate": 0.001,
    "traffic": "project/file/traffic_15_nodes_net.json"
}

```

Listing 5.9: Example of sim_params.json file

The others file, such as *traffic.json* and file concerning SeQUeNCe and NetworkX graphs, have been explained in the previous chapters.

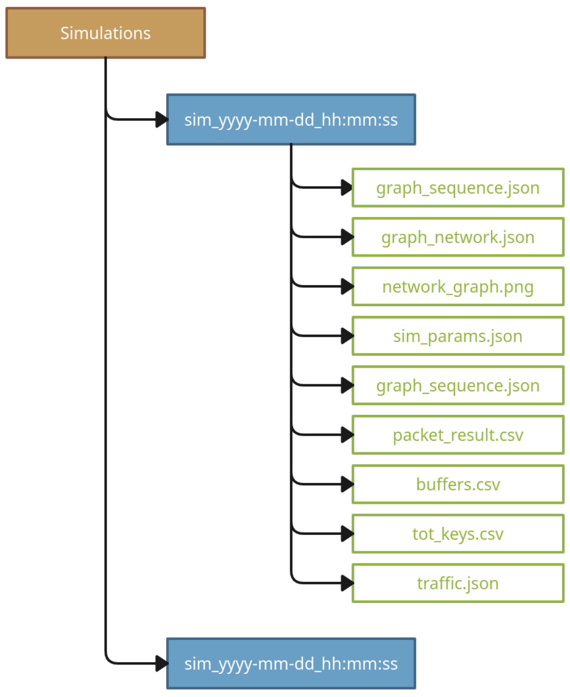

Figure 5.5: Structure of the simulation folder.

Figure 5.6 provides a high-level view of the various explained components of the extension.

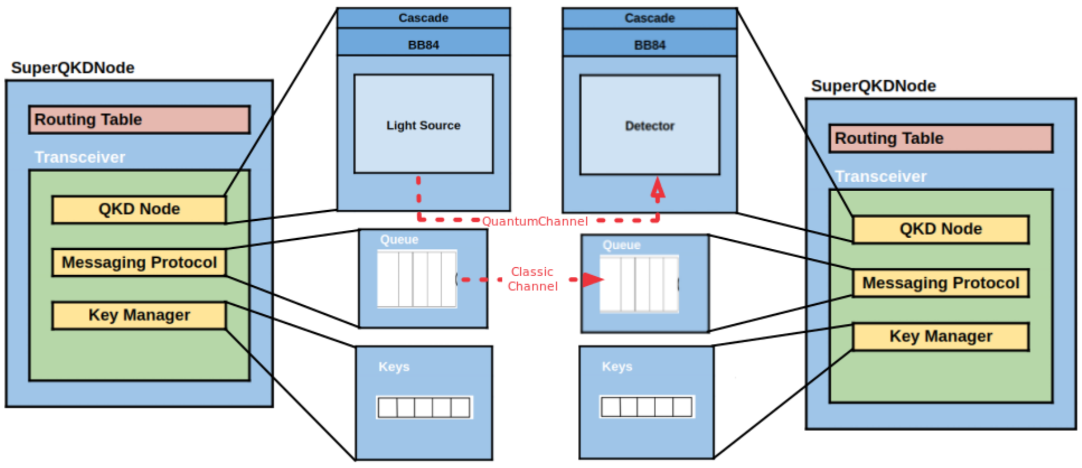

Figure 5.6: High-level view of Sequence extension components.

Chapter 6

Evaluation

In this chapter we will run and analyze the various simulations on QKD networks with FSO Link, but before going on to run and analyze the simulations we will see an introductory part to analyze the coherence of the extension.

6.1 Extension Consistency

To analyze the consistency of the extension we are going to perform simulations with the aim of seeing its behavior. As a first analysis we are going to carry out simulations in which we send a sequence of bits with the BB84 protocol on which the Cascade protocol will act.

6.1.1 Bit Error Rate analysis

We will simulate on links with a fixed bit rate of the quantum channel at **1 Mbps** as the BER varies, to verify the number of bits obtained from the receiver. The simulations were not conducted using the extension directly but rather through the use of SeQUeNCe and the BB84 and Cascade protocols. A very important variable, within the component responsible for the emission of photons, is the `mean_photon_number`, which follows a Poisson distribution with a customizable mean. This variable indicates the mean number of photons emitted each period. This variable will therefore influence the number of photons sent, in fact it may be that in a period a photon is not sent thus generating losses.

For example, if we want to send a certain number of qubits and we have a `mean_photon_number` of 0.1, per period, we will send a number of qubits determined by the result of the Poisson distribution with a mean of 0.1. So, it's possible that in one period, we may send zero qubits, and in another, one or more qubits. This, of course, will result in losses. However, we are interested in perfect emission, meaning that one and only one qubit is sent in each period. To achieve this, the code in the SeQUeNCe source code related to photon emission has been modified, forcing the sending of only one qubit per period. The modification was made within the LightSource class in the `emit` method, Listing 6.1 shows this modification. So, in the next chapters, we will see results coming from simulations with this modification.

```
    time = self.timeline.now()
    period = int(round(1e12 / self.frequency))

    for i, state in enumerate(state_list):
        # num_photons = self.get_generator().poisson(self.mean_photon_num) (
    removed)
        num_photons = 1 # new value

        ...

        for _ in range(num_photons):
            # schedule emission (unchanged)

        time += period
```

Listing 6.1: Modification on LightSource class in emit method

So we will simulate on 4 different BERs, **0%**, **25%**, **50%** and **75%**. For each BER we are going to perform 30 simulations in which A, through BB84, sends 10'000 bits and subsequently the error correction is applied on the bits with Cascade. Figure 6.2 shows these results. Figure 6.1 outlines with a block diagram the process we will analyze. There is a first phase involving sending bits from the source node to the destination node using the BB84 protocol, a second phase involving the Sifting stage, and a third and final phase where the Cascade protocol is applied to the bits obtained after the Sifting phase.

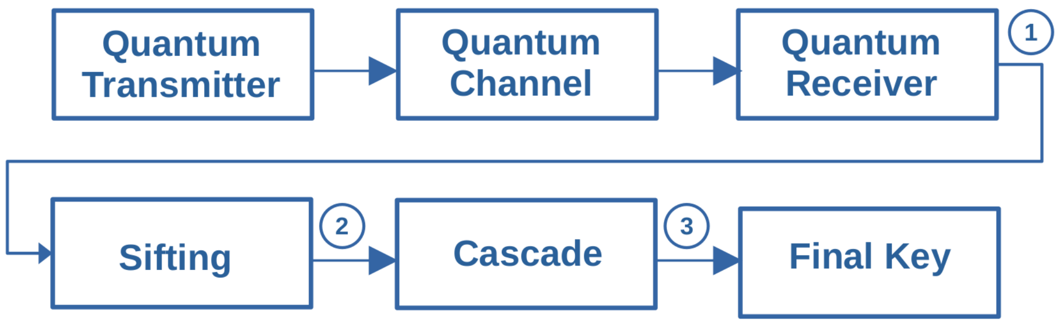

Figure 6.1: Descriptive diagram of the Sifting and Cascade process.

As can be seen from Figure 6.2, for each BER we are going to run 30 simulations, each of which sends 10,000 bits. In the first box plot on the top left we see that the blocks are flat, in fact it is correct because we have that for each period we send exactly one bit, and since we have not set parameters to simulate losses, we see that before the sifting phase the bits sent they are always and exactly 10,000. Obviously in this phase the BER is not influential. So this meets our predictions. So, after this step, the receiving node has all the bits sent by the source node without doing the sifting phase. This is at Point 1 in Figure 6.1, the end of sending and receiving bits.

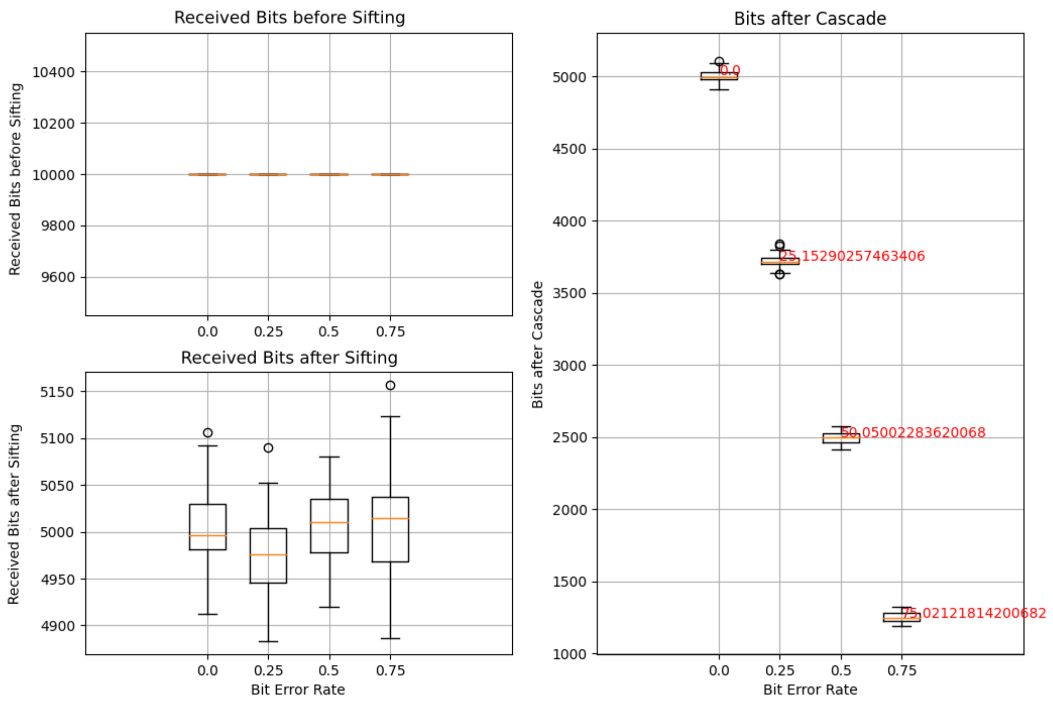

Figure 6.2: Result of simulations with varying BER.

The box plot at the bottom left shows the number of bits obtained after the sifting phase. We are at Point 2 in Figure 6.1. During the Sifting phase, as explained in the previous chapters, approximately half of the bits are discarded following the exchange of bases. In fact, what we observe in the graph is that on average we see that approximately half of the bits are discarded.

Finally, in the box plot that occupies the right part of the image, we see the number of bits obtained after executing the Cascade protocol. Finally we are at Point 3 in Figure 6.1. As described in previous chapters, Cascade, based on BER, will reveal bits for error correction, these bits will then have to be removed from the key once leaked, so based on BER we expect to see a percentage gap of bits as the BER percentage. In fact, in the graph we see that on average the number of bits discarded in percentage is like the BER. So if we have a BER of 50% we expect about 50% of the key bits to be discarded.

6.1.2 Key Rate and Link analysis

Once we have seen the consistency of the BER, let's now look at the Key rate and its consequences on the classic channel, for the exchange of messages, as a second analysis. As with the previous simulations, in this case too we are sending exactly one photon per period, so as not to have losses.

As explained in the previous chapters, the extension uses a model for determining the BER through mainly two parameters: the distance and the bit rate of the channel. So to verify the results on different BERs we will perform 4 simulations, each based on links of different lengths, exactly on lengths of 2000, 3000, 4000 and 5000 meters, in order to estimate a key rate.

Therefore each simulation will have the following characteristics (Figure 6.3 summarizes the results obtained):

- Simulation time 1 second
- Link capacity 1 Mbps
- Key length 128 bits
- Lengths ranging from 2000 to 5000 meters, with steps of 1000 meters

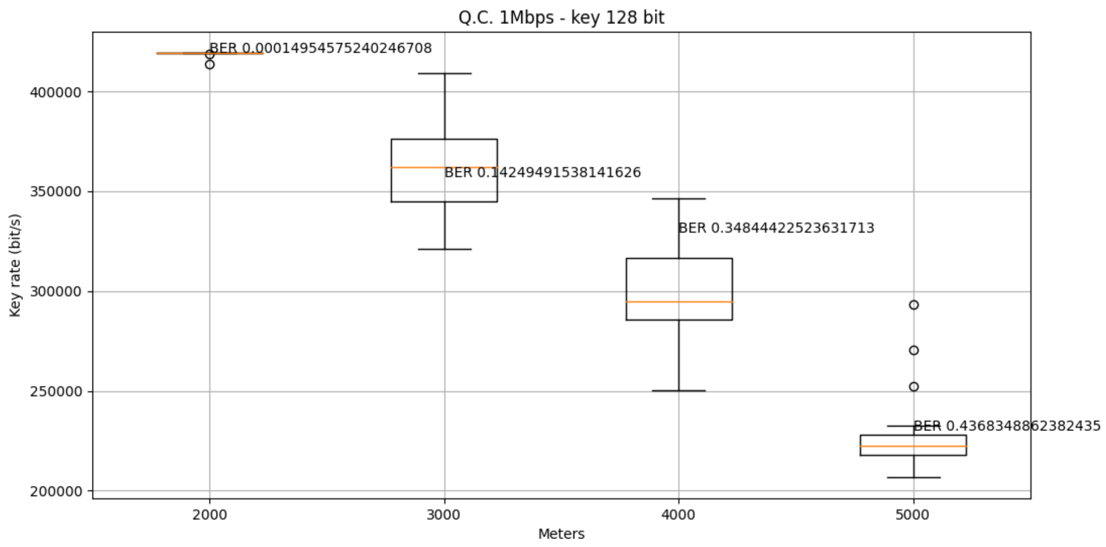

Figure 6.3: Result of simulations for Key Rate.

In Figure 6.3 we see the results after the Sifting phase. We see that with a distance equal to 2000 meters we have a key rate of approximately 420,000 bits, with a BER of approximately 0%. Considering that the maximum capacity of the link is 1 Mbps and the simulation time is 1 second, we expect 1,000,000 bits to be transmitted in the link and then approximately half depending on the sifting phase. In fact, we observe this behavior in the graph. With a BER of about 0% we have that the available bits are approximately half of 1,000,000 bits.

Subsequently, taking the maximum available bit value for 2000 meters as the value, we see that the BER successes are also coherent. In fact we observe that with a BER of approximately 43% the available bits are approximately 43% of 420'000 bits.

Next we analyze how the classic channel behaves. We expect that the classical channel is limited by the quantum channel, in fact, as explained in the previous chapters, sending a message depends or not on the presence of keys. So if the quantum channel is slower than the classical channel we expect it to take on the role of a bottleneck.

To see this behavior we are going to run some simulations in the following way: we are going to run some simulations at various distances, each of these simulations will have an increasing load, what we expect to see is that at a certain point the number of bits sent will be limited, but not to the maximum capacity of the classic link, so it is not saturated. In reality we are seeing that it is the quantum link that is saturated so the sending of bits in the classical channel will be limited by the quantum channel. In Figure 6.4 we see the results of these simulations.

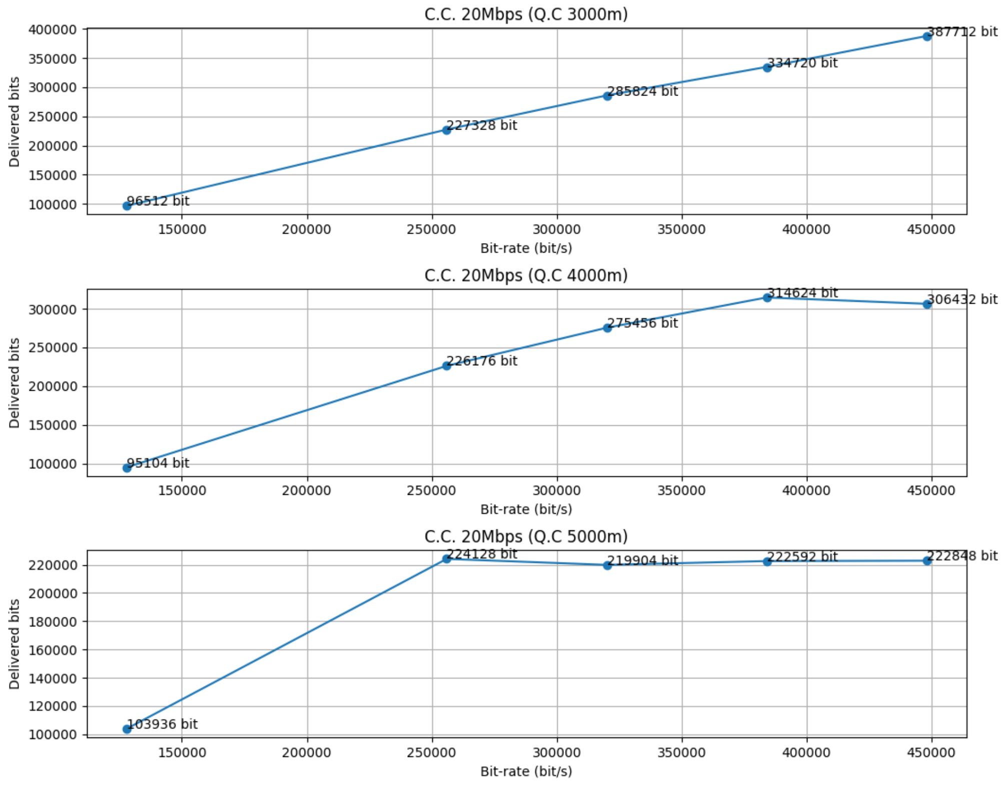

Figure 6.4: Simulation results for increasing load.

In the initial graph, at the top, of Figure 6.4 we see the results of the simulations performed on classical links of 20 Mbps, as indicated in the title of the graph, in parallel the QKD takes place on a quantum channel (FSO) with distance of 3000 meters and a fixed bit rate of 1 Mbps.

We see that there is an increase in the load, i.e. in the bits sent per second, we can also see that in this graph we do not see a limitation, rather we see that as the load increases the bits sent also increase, this is because with these loads we do not reach the saturation of the quantum link. An estimate of the key rate for these simulations is the value for the distance of 3000 meters in Figure 6.3, which is approximately 370'000 bits.

In the central graph of Figure 6.4 we see the result of the simulations always on a classic link of 20 Mbps while in parallel a QKD is performed on a quantum channel with a distance of 4000 meters and a bit rate of 1 Mbps, an estimate of the key rate for these simulations is given from the value for 4000 meters in Figure 6.3, i.e. approximately 300'000 bits. In fact we see that at a certain point as the load increases we see a flattening, this means that we have reached the maximum key rate, therefore the classical channel will not saturate and will be limited by the quantum one. Analogously this is observed for the bottom graph of Figure 6.4.

6.2 Simulations Evaluation

6.2.1 Simulation on 10-nodes chain Network

We will now conduct two type simulations to observe the behavior of a 10-node chain network under both low and heavy loads. Two networks will be constructed, each comprising 10 nodes in a chain. The key distinction lies in the distances between the nodes, with one network having a node-to-node distance of 1000 meters and the other 4000 meters. This ensures a lower Bit Error Rate (BER) for the former, thereby resulting in a greater number of keys during the simulation. The traffic pattern, defined in a JSON file, dictates that all nodes exchange messages with each other, excluding self-interactions. In both configurations, each channel will have a bit rate of 1 Mbps. We can see in Listings 6.2 and 6.3 the configuration of the two networks, in Listing 6.4 the traffic configuration for both.

```
{
    "nodes": [
        {
            "name": "node0",
            "type": "QKDNode"
        },
        ...
        {
            "name": "node9",
            "type": "QKDNode"
        }
    ],
    "qchannels": [
        {
            "name": "qchannel0_0to1",
            "source": "node0",
            "destination": "node1",
            "distance": 1000,
            "bit_rate" : 1000000
        },
        ...
    ],
    "cchannels": [
        {
            "name": "cchannel0_0to1",
            "source": "node0",
            "destination": "node1",
            "distance": 1000,
            "bit_rate" : 20000000
        },
        ...
    ]
}
```

Listing 6.2: Configuration of a 10-node network with link lengths of 1000 meters.

```
{
    "nodes": [
        {
            "name": "node0",
            "type": "QKDNode"
        },
        ...
        {
            "name": "node9",
            "type": "QKDNode"
        }
    ],
    "qchannels": [
        {
            "name": "qchannel0_0to1",
            "source": "node0",
            "destination": "node1",
            "distance": 4000,
            "bit_rate" : 1000000
        },
        ...
    ],
    "cchannels": [
        {
            "name": "cchannel0_0to1",
            "source": "node0",
            "destination": "node1",
            "distance": 1000,
            "bit_rate" : 20000000
        },
        ...
    ]
}
```

Listing 6.3: Configuration of a 10-node network with link lengths of 4000 meters.

```
{
    "packets" : [

        {
            "src_node" : "node0",
            "dst_node" : ["node1","node2","node3","node4","node5",
                          "node6","node7","node8","node9"]
        },
        {
            "src_node" : "node1",
            "dst_node" : ["node0","node2","node3","node4","node5","node6",
                          "node7","node8","node9"]
        },
        {
            "src_node" : "node2",
            "dst_node" : ["node0","node1","node3","node4","node5","node6",
                          "node7","node8","node9"]
        },
        {
            "src_node" : "node3",
            "dst_node" : ["node0","node1","node2","node4","node5","node6",
                          "node7","node8","node9"]
        },
        {
            "src_node" : "node4",
            "dst_node" : ["node0","node1","node2","node3","node5","node6",
                          "node7","node8","node9"]
        },
        {
            "src_node" : "node5",
            "dst_node" : ["node0","node1","node2","node3","node4","node6",
                          "node7","node8","node9"]
        },
        {
            "src_node" : "node6",
            "dst_node" : ["node0","node1","node2","node3","node4","node5",
                          "node7","node8","node9"]
        },
        {
            "src_node" : "node7",
            "dst_node" : ["node0","node1","node2","node3","node4","node5",
                          "node6","node8","node9"]
        },
        {
            "src_node" : "node8",
            "dst_node" : ["node0","node1","node2","node3","node4","node5",
                          "node6","node7","node9"]
        },
        {
            "src_node" : "node9",
            "dst_node" : ["node0","node1","node2","node3","node4","node5",
                          "node6","node7","node8"]
        }
    ]
}
```

Listing 6.4: Traffic for 10-nodes network

One of our focus during the simulations is on observing the number of transmitted bits on each classical link. Our expectation is that, under a low load, only the central links of the chain will become saturated. Conversely, with a heavy load, we anticipate an undefined behavior where more links become saturated. We will perform 30 simulations under low load and 30 under heavy load on the network. The 30 simulations for each load configuration will differ due to random seeds, as they are modeled by random variables. Each run will be executed under different seeds. The simulation parameters for low load are as follows:

- The duration of a single simulation is 1 second.
- Message: 25 messages per second on average ($1/25 = 0.04$ interdeparture time).
- Key size 128 bits.
- Packet size 128 bits.

While the simulation parameters for heavy load are as follows:

- The duration of a single simulation is 1 second.
- Message: 500 messages per second on average ($1/500 = 0.002$ interdeparture time).
- Key size 128 bits.
- Packet size 128 bits.

The first simulations result for a low load, illustrated in Figure 6.5, reveal a bell-shaped distribution of the graph. In the graph, we are observing the number of bits transmitted by each link. On the x-axis, we see the link number, from 1 to 9. On the y-axis, we see the number of bits transmitted by the corresponding path on the x-axis. This visually indicates that the saturated links are predominantly the central ones. Notably, the network configuration involves a 1000-meter distance between links, resulting in a zero BER.

The outcomes for heavy load are also depicted in Figure 6.5. In this instance, the links that become saturated are not solely the central ones; the distribution takes on a bimodal shape. Additionally, we observe that the saturated links follow the maximum key rate for networks configured with a 4000-meter distance between links, as illustrated in Figure 6.3.

The results we are interested in analyzing include, for each destination node, the number of bits successfully received in both low load and heavy load configurations. The simulation parameters remain the same as in the previous simulations, as the data analyzed is derived from those same simulations. Therefore, the results are based on 30 simulations for low load and 30 for heavy load. In Figure 6.6, we observe the results. On the x-axis, we have the

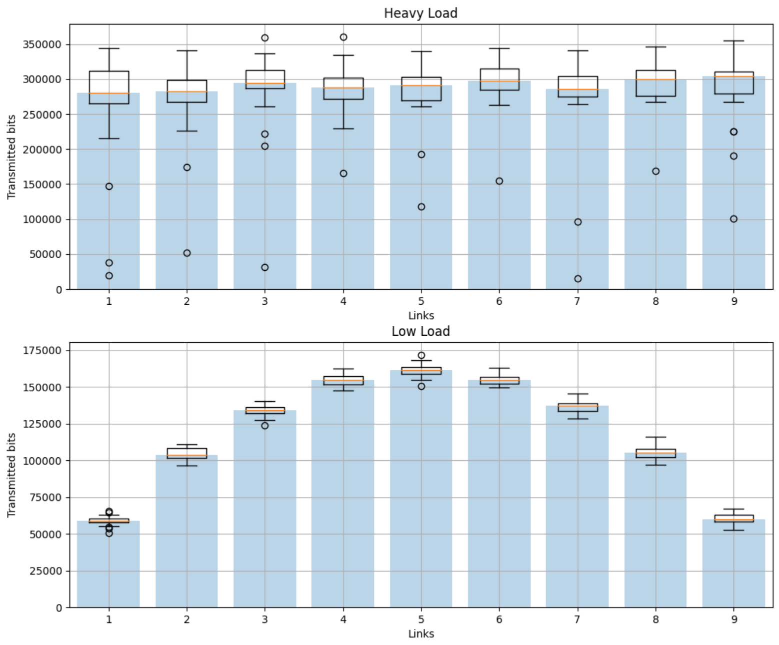

Figure 6.5: Simulation results increasing load.

nodes, and on the y-axis, we have the number of bits successfully delivered for the respective node. In the upper part, we see the results for heavy load, noting that graphically it forms a kind of parabola. Thus, we observe that central nodes receive fewer bits than peripheral nodes in the network, which is due to the saturation of central nodes occurring earlier than in peripheral nodes, leading central nodes to receive fewer bits. Additionally, we expect that in heavy load, longer paths have a lower success probability, as we will analyze in the subsequent results. Meanwhile, in the lower part of Figure 6.6, we see that the bits received by each destination node are approximately similar, as the network is not under heavy load, and even packets with a longer path have a higher success probability.

In order to better visualize the number of bits received by each destination node, we display the heatmap in Figure 6.7, generated from the previous simulations. As can be observed, for heavy load and paths longer than 2, the received bits decrease drastically, indicating a low success probability for long paths. In the right part of Figure 6.7, the heatmap for low load is presented, showing a more uniform distribution. However, it's noticeable that longer paths receive fewer bits. Figure 6.8 and Figure 6.9 provides a more explicit representation of the distributions for heavy load and low load.

Figure 6.8 shows the average number of bits received by destination nodes for paths of length from 1 to 9, meaning from the smallest simple path to the longest existing simple path. Additionally, the second y-axis shows the number of paths of a certain length present

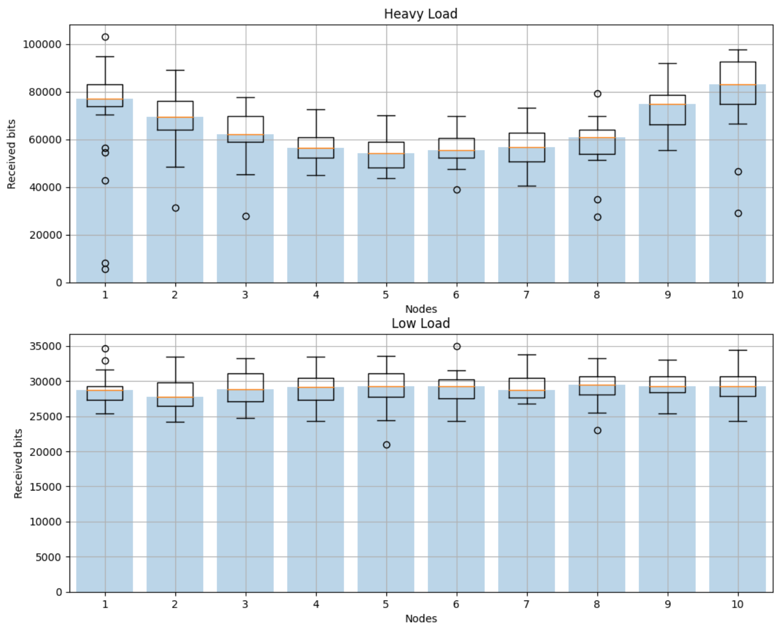

Figure 6.6: Result of bits received by each destination node.

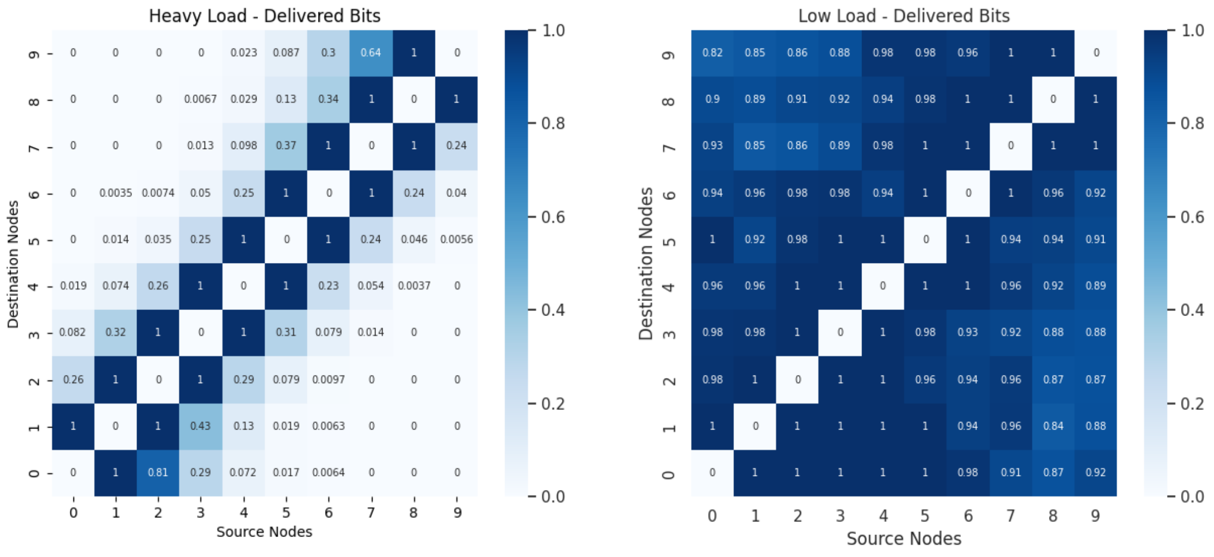

Figure 6.7: Bits successfully delivered by source node and destination node.

in the network. In the top part, we see the results for a Heavy Load configuration, where the average number of bits decreases as the path length increases. This is the expected

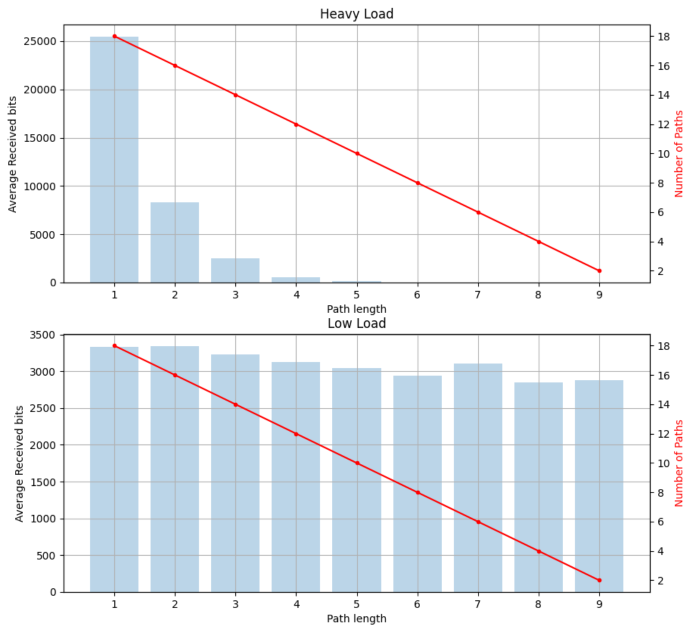

Figure 6.8: Distribution of successfully delivered bits by path length.

behavior for a Heavy Load configuration. In the bottom part of the image, we find the results for the Low Load configuration, where we notice that in this case, the number of bits slightly decreases with the increasing length of the paths compared to a Heavy Load configuration, as the central nodes have not reached saturation. For both configurations, since the network is the same, the number of paths of a certain length will be the same. Being a chain, we see that the number of paths of length 1 is the highest, and as the length increases, they decrease linearly.

In Figure 6.9, we have a representation of success probabilities for paths of different lengths, from 1 to 9. On the x-axis, we have the path lengths, while on the y-axis, we have the success probability. In the top part of the image, we see the results for the Heavy Load configuration, where longer paths equal to or greater than 4 have an almost zero success probability. The probabilities are calculated as the fraction between the average of sent bits and the average of correctly delivered bits for each path length. In the bottom part of the figure, we find the results for the Low Load configuration. In this case, we notice that all paths of length from 1 to 9 have a success probability greater than 80%.

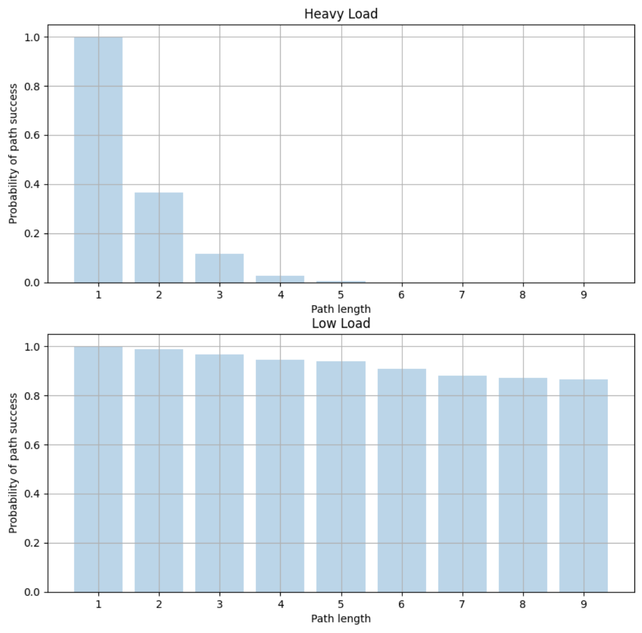

Figure 6.9: Probability of successfully delivered bits by path length.

6.2.2 Simulation on a connected 15-nodes Network

In this section, we will perform a simulation on a random network of 15 connected nodes created using NetworkX and parsed by our parser to be executed within our extension. In this simulation, similar to the previous ones, we will analyze the amount of bits received by each node from every other node in the network, the success probability of paths of different lengths, and the average bits received by each node. This allows us to study the saturation points of the network and its overall behavior.

We begin with the generation of the network. The network is generated using the NetworkX method `random_internet_as_graph(num_nodes)`, which will provide us with an object representing a connected network of 15 nodes. Through the extension, we will generate an image of the network and a JSON file representing the network in a format suitable for sequences. The JSON file will initially return default values for certain parameters, which will later be modified with appropriate values for the simulation before its initiation. The Figure 6.10 depicts the randomly generated network that will be used for the simulation.

The length of the links between the nodes in the network will be 1000 meters, both for quantum and classical channels. Additionally, the bit rate for quantum channels will be 1 Mbps, while for classical channels, it will be 20 Mbps. We are interested in ensuring that during the simulation, each node receives at least 50 packets from every other individual node. Therefore, the simulation parameters are as follows:

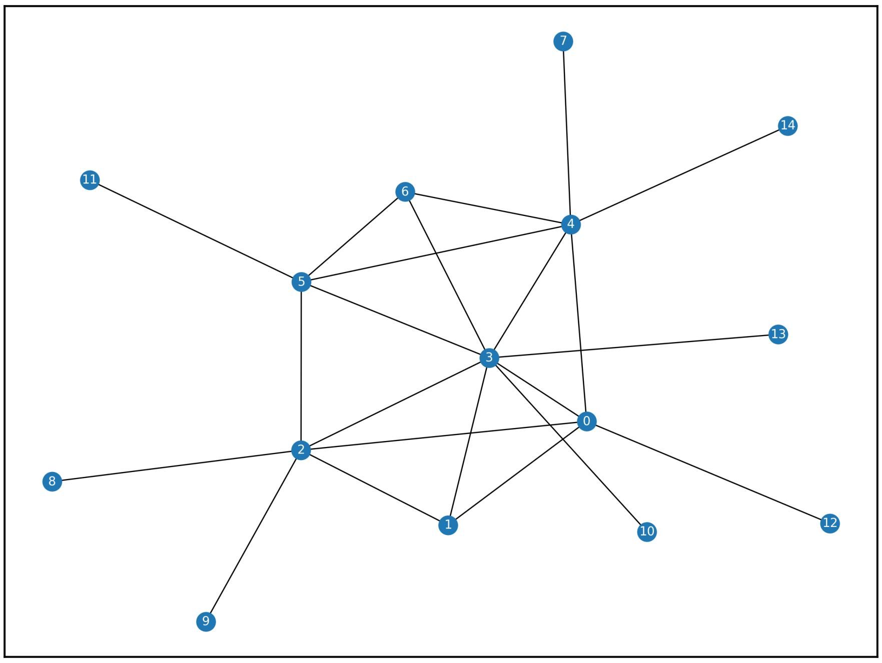

Figure 6.10: Randomly generated network used for the simulation

- Key size: 128 bit
- Packet size: 128 bit
- Message: 100 messages per second on average (1/100 = 0.01 interdeparture time).
- Duration: $15(nodes) \times 50(num.packet) \times 0.01(Mess.rate) = 7.5$ *seconds*

The simulation command, once the parameter setup is complete, is as follows (the packet and key size are default parameters set to 128 bits in the extension):

```
python3 project/sim_ext.py --sim-time 7.5 --seq-graph project/file/
    graph_15_nodes.json --traffic project/file/traffic_15_nodes_net.json --
    inspection-rate 0.001 --mess-rate 0.01
```

Listing 6.5: Command for run the simulation

As a first result, we will analyze the heatmap representing the fraction of bits received by each node from all other nodes in the network, shown in Figure 6.11.

As we can see from Figure 6.11, each node in the network correctly receives all the bits sent by the source node. As most values in the matrix, except for the diagonal, have a value of 1, this means that the sent bits are equal to the bits received correctly. These values are indeed a fraction between the received bits and the sent bits, indicating the percentage of bits received compared to the total. The columns represent the source nodes, while the rows

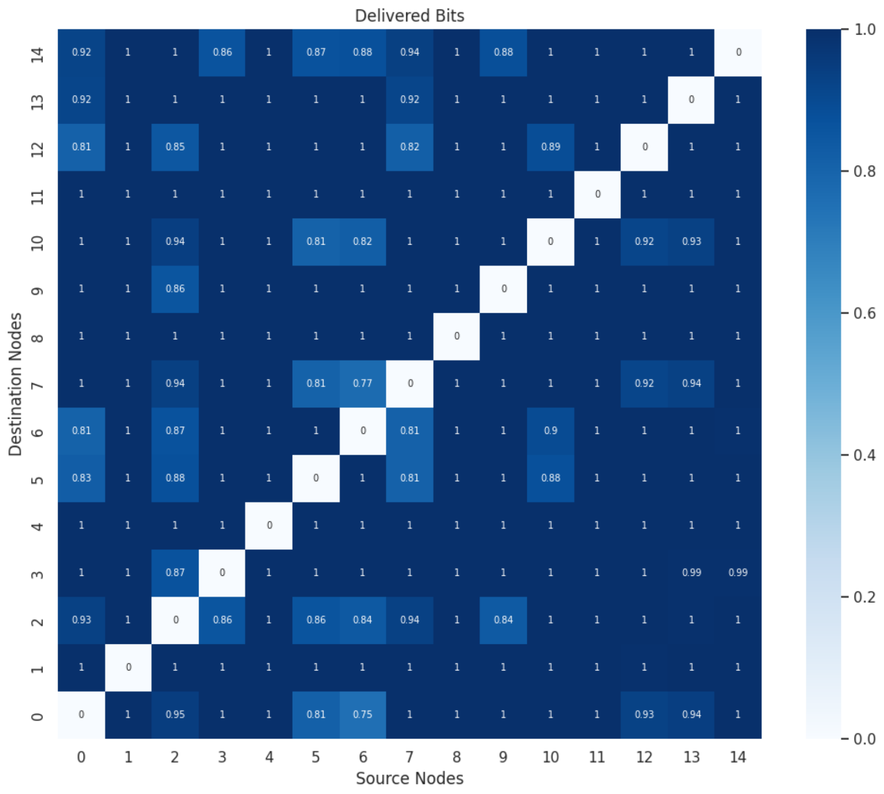

Figure 6.11: The resulting heatmap after the simulation

represent the destination nodes. The central diagonal will have a zero value as there are no message transmissions from a node to itself. From the distribution of bits, we observe that there are no nodes receiving a significantly lower amount of bits. Therefore, for the specified parameters, we are in a low-load configuration.

Subsequently, we analyze, for each node, the number of bits received correctly from every other node in the network. This provides a different perspective compared to the heatmap. In Figure 6.12, we see the total bits received for each destination node. As we can observe, the number of bits received by each node is roughly the same. This is because we are in a Low load configuration, and thus, there are no central nodes reaching saturation, resulting in a lower probability of success for longer paths.

Now, let's analyze the average of correctly received bits by destination nodes for paths of different lengths. For each path length x, from node A to node B, we calculate how many bits the destination node has correctly received and then take the average. We repeat this process for all paths of lengths from 1 to L, where L is the length of the longest path. In Figure 6.13, we see the results for the simulation. On the left y-axis of the graph, we indicate

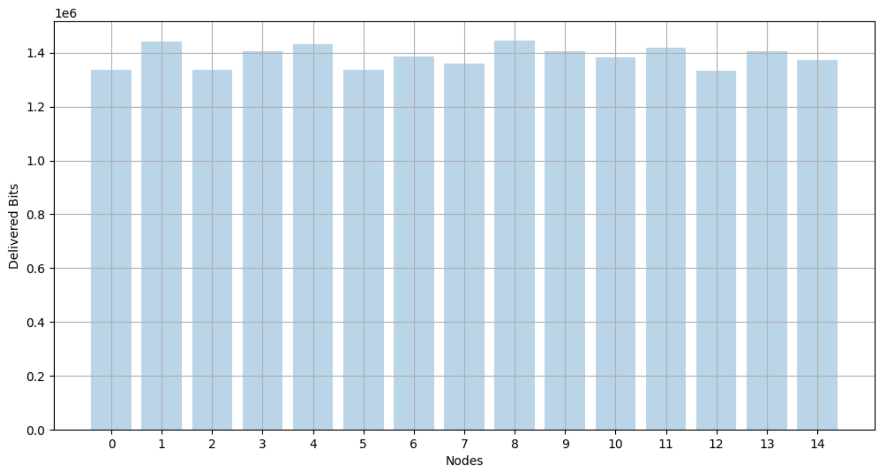

Figure 6.12: Received bits for each node

the average number of bits correctly received for each path length from 1 to L. We observe that for our network of 15 nodes, there are simple paths (no cycles) with a maximum length of 8. Meanwhile, on the right y-axis, we indicate the number of paths of a certain length x. We can observe that, on average, the number of bits correctly received by each destination node on paths of lengths from 1 to 8 is approximately the same.

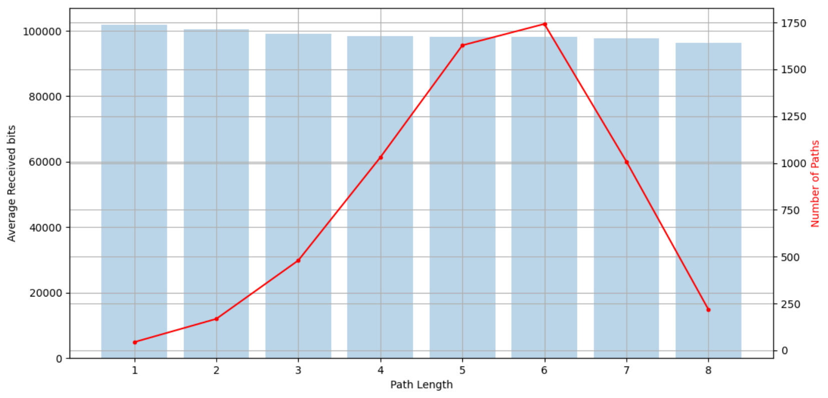

Figure 6.13: Average received bits for each node

As a final result, let's analyze the success probability of each path length. In Figure 6.14, we can see the results for the simulation. Being in a low-load configuration, we observe that the success probabilities for each length are very high. This is because the number of packets sent by each node to various destinations, even for distant destinations, is approximately equal to the number of packets received correctly.

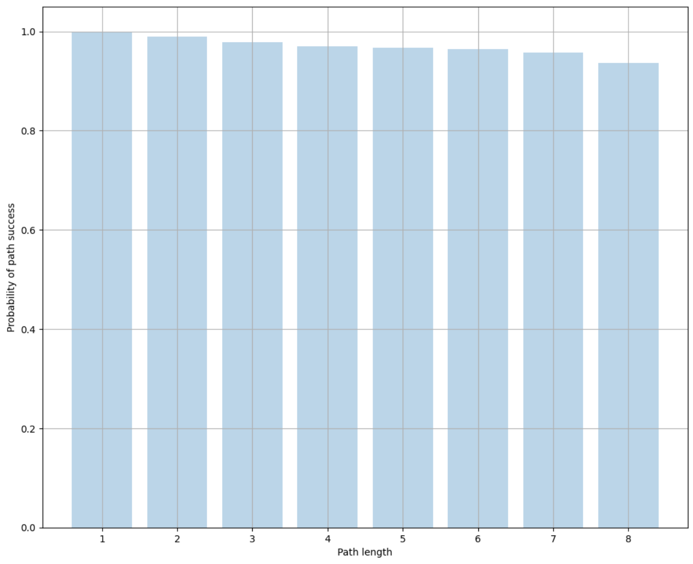

Figure 6.14: Success probability of paths

Figure 6.15, instead, shows us the centrality of nodes within the network. We calculate the centrality of each node using the NetworkX function called `betweenness_centrality`. Betweenness centrality is a measure of centrality in a graph based on the shortest paths. For each pair of vertices in a connected graph, there is at least one shortest path between them. The betweenness centrality for each vertex is the number of these shortest paths that pass through the vertex.

As we can see from Figure 6.15, we have the centrality values for each node. On the x-axis, we have the names of the nodes sorted based on their centrality values, which are on the y-axis. Notice that Node 3 has a higher centrality value than all the others, followed by nodes 4, 2, 5, and 0. The remaining nodes have a centrality value of zero.

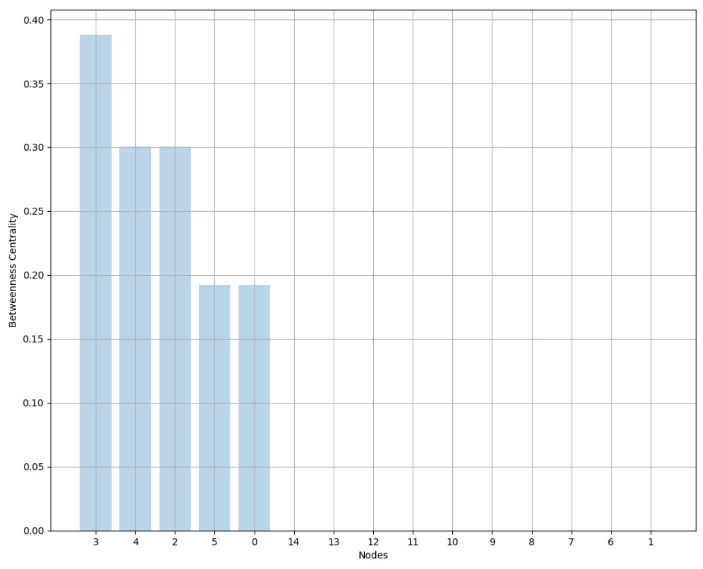

Figure 6.15: Centrality of the nodes.

Now, let's analyze the results for a simulation with a Heavy Load configuration on the same network, as shown in Figure 6.10. We will be examining the same data as in the previous simulation. The configuration parameters for this simulation are the follows:

- Key size: 128 bit
- Packet size: 128 bit
- Message: 500 messages per second on average ($1/500 = 0.002$ interdeparture time).
- Duration: $15(nodes) \times 50(num.packet) \times 0.002(Mess.rate) = 1.5\ seconds$

In addition to increasing the load by increasing the messages each node has to send, we also changed the length of the quantum links (FSO) in the network topology to increase the Bit Error Rate (BER). As mentioned in the introduction of this thesis, simulations with heavy loads could saturate the computer's RAM, so by increasing the BER, we limit the key generation. The length of the quantum links has been set to 4000 meters with a Bit rate of 1 Mbps. Meanwhile, the length of the classical links remains unchanged at 1 meter, with a Bit rate of 20 Mbps.

The simulation command, once the parameter setup is complete, is as follows (the packet and key size are default parameters set to 128 bits in the extension):

```
python3 project/sim_ext.py --sim-time 1.5 --seq-graph project/file/
    graph_15_nodes.json --traffic project/file/traffic_15_nodes_net.json --
    inspection-rate 0.001 --mess-rate 0.002
```

Listing 6.6: Command for run the simulation

As a first result, let's analyze the heatmap in Figure 6.16. On the x-axis, we have the source nodes, ranging from 0 to 14. On the y-axis, we find the destination nodes, also ranging from 0 to 14. In the various cells, we have the fraction of correctly received bits compared to the bits sent for each source-destination pair. Unlike the heatmap for the Low Load configuration, here we observe a significant difference. In fact, most nodes receive a percentage of bits between 0% and 50% compared to those sent. We can better understand these values by looking at Figure 6.17, which indicates the total number of bits each node has correctly received on the y-axis. As we observe in Figure 6.17, nodes 0 to 7 receive fewer bits because, as seen in Figure 6.10 of the topology, they are central nodes that have a higher incoming bit load, leading to an accumulation of packets to forward, filling the queues, and dropping various messages to send.

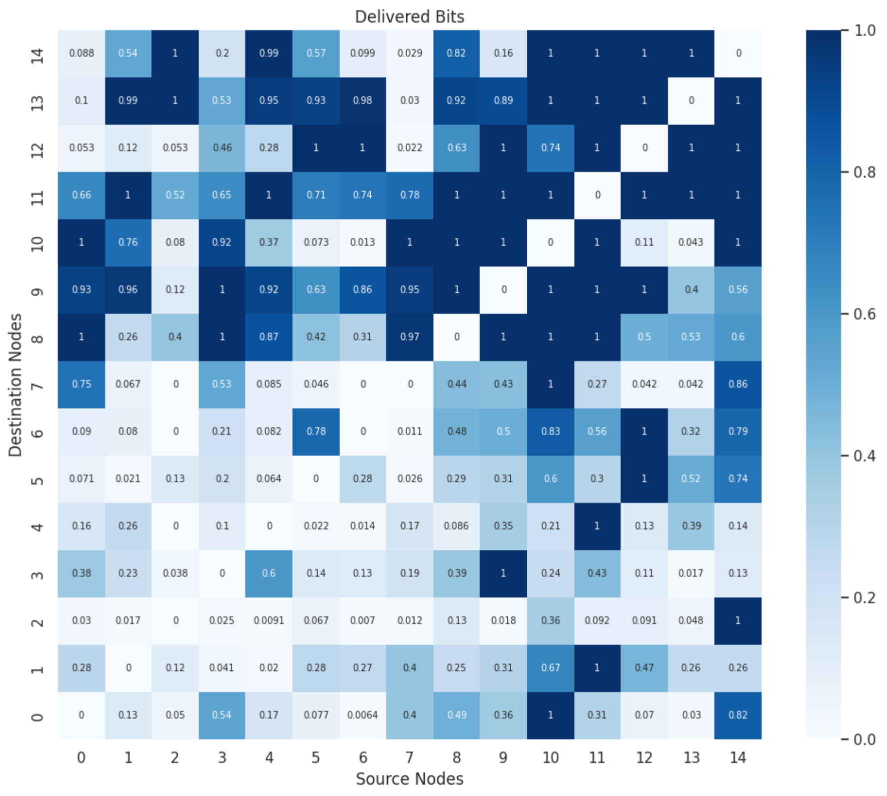

Figure 6.16: Fraction of bits delivered correctly.

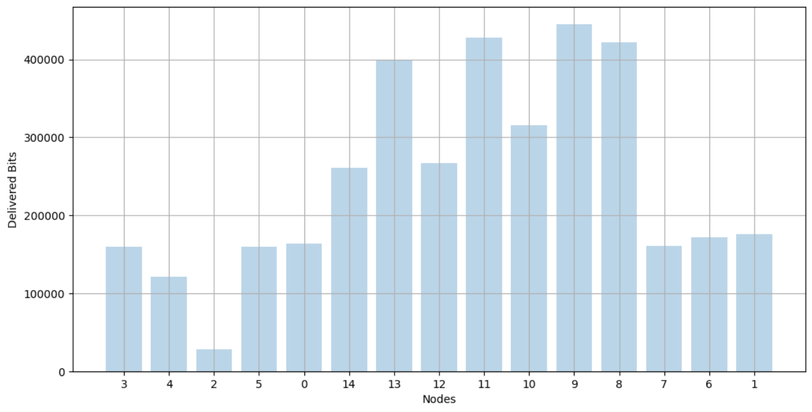

Figure 6.17: Bits delivered correctly per nodes. Sorted for node centrality.

Let's now look at the distribution of correctly delivered bits by path length and the success probability for each length, similar to the analysis for Low Load. In Figure 6.18, we observe the quantity of correctly delivered bits to each destination node on paths of length 1 to 8. On the x-axis, we have the length of the various paths. On the left y-axis, we find the average number of bits delivered correctly, and on the right y-axis, we find the number of paths for a given length. It can be noted that we are in a heavy load configuration as the quantity of correctly delivered bits decreases with the increase in path length. This is because it is more likely to encounter more congested nodes in longer paths. We observe this behavior in Figure 6.19 as well, representing the success probability of path lengths. On the x-axis, we have the various path lengths, while on the y-axis, we find the success probability. We notice that as the lengths increase, the success probability decreases, indicating again that we are in a Heavy Load configuration.

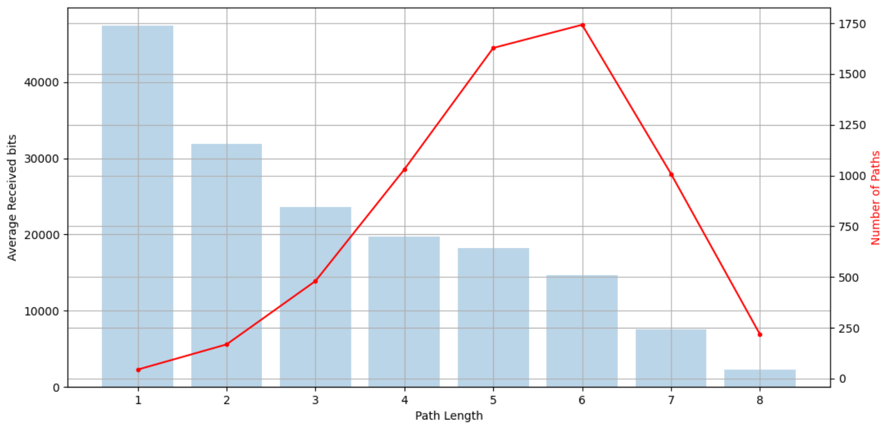

Figure 6.18: Average received bits for each node.

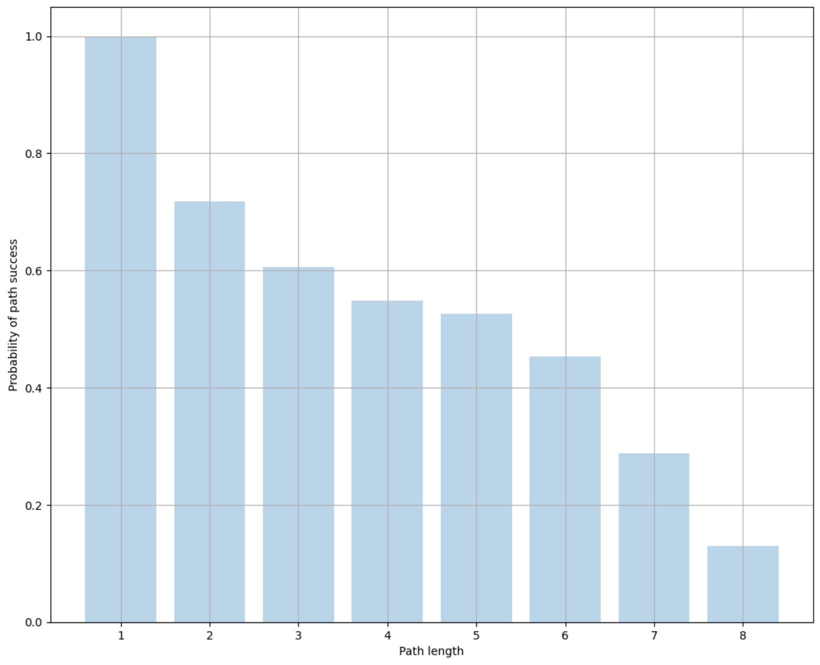

Figure 6.19: Success probability of paths.

Chapter 7

Conclusion

In this thesis, we have provided an implementation of an extension for the Sequence simulator [18] to simulate networks implemented with Free Space Optic (FSO) Links and analyze their performance. As discussed at the beginning of this thesis, Quantum Key Distribution (QKD) will be fundamental in the near future as quantum computing becomes more accessible, rendering current cryptography less effective. During the thesis, our focus was on implementing the extension, conducting various simulations and analyses to test its correctness and reliability. The initial part of the implementation involved modeling the Free-Space Optical (FSO) links and integrating a model for calculating the Bit Error Rate (BER), developed by Leonardo Maccari[1] and Peppino Fazio[2] [26]. Once we achieved a certain reliability, we shifted our attention to simulations on Quantum Key Distribution (QKD) networks. These simulations were performed using our extension and its components. An interesting addition to the work could be implementing dynamic routing based on link saturation or the available keys of a node. Additionally, with suitable hardware, conducting simulations on real hardware to create a trusted relay-based QKD network.

[1]Department of Environmental Sciences, Informatics and Statistics, Ca' Foscari University of Venice, Italy

[2]Department of Molecular Sciences and Nanosystems, Ca' Foscari University of Venice, Italy

List of Figures

Listings

Bibliography

[1] Xiaoliang Wu and Alexander Kolar and Joaquin Chung and Dong Jin and Tian Zhong and Rajkumar Kettimuthu and Martin Suchara, *"SeQUeNCe: A Customizable Discrete-Event Simulator of Quantum Networks"*, arXiv, 2020.

[2] Peter Shor, *"Algorithms for quantum computation: Discrete log and factoring"*, in Proceedings of the 35th Annual Symposium on the Foundations of Computer Science, Santa Fe, IEEE Computer Society Press,1994.

[3] Takaaki Matsuo, *"Simulation of a Dynamic, RuleSet-based Quantum Network"*, Keio University,2019.

[4] K Shannon, E Towe, and O Tonguz, *"On the Use of Quantum Entanglement in Secure Communications: A Survey"*, arXiv:2003.07907v1 [cs.CR] 17 Mar 2020.

[5] R.Alleaumea, C.Branciard, J.Bouda, T.Debuisschert, M.Dianati, N.Gisin, M.Godfrey, P.Grangier, T.Langer, N.Lutkenhaus, C.Monyk, P.Painchault, M.Peev, A.Poppe, T.Pornin, J.Rarity, R.Renner, G.Ribordy, M.Riguidel, L.Salvail, A.Shields, H.Weinfurter, A.Zeilinger, *"Using quantum key distribution for cryptographic purposes: Asurvey"*, http://dx.doi.org/10.1016/j.tcs.2014.09.018,2014.

[6] W. Diffie, M.E. Hellman, *"New directions in cryptography"*, IEEE Trans. Inform. Theory 22 (1976) 644-654.

[7] Bennett, C. H., and G. Brassard, *"Quantum Cryptography: Public Key Distribution and Coin Tossing"*, Proc. of IEEE Int. Conf. on Comput. Sys. and Sign. Proces." (1984).

[8] Stebila, D., Mosca, M., and Lutkenhaus, N., *"The case for quantum key distribution"*, In Quantum Communication and Quantum Networking: First International Conference, QuantumComm 2009, Naples, Italy, October 26-30, 2009, Revised Selected Papers 1 (pp. 283-296). Springer Berlin Heidelberg.

[9] Yuan Cao , Yongli Zhao , Qin Wang , Jie Zhang, Soon Xin Ng and Lajos Hanzo , *"The Evolution of Quantum Key Distribution Networks: On the Road to the Qinternet"*, IEEE COMMUNICATIONS SURVEYS & TUTORIALS, VOL. 24, NO. 2, SECOND QUARTER 2022

[10] Tang, X., Wonfor, A., Kumar, R., Penty, R. V. and White, I. H., *"Quantum safe metro network with low-latency reconfigurable quantum key distribution"*, J. Lightw. Technol., vol. 36, no. 22, pp. 5230-5236, Nov. 15, 2018.

[11] Y.-L. Tang et al., *"Measurement-device-independent quantum key distribution over untrustful metropolitan network"*, Phys. Rev. X, vol. 6, no. 1, Mar. 2016, Art. no. 011024.

[12] Chip Elliott, *"The DARPA Quantum Network"*, arXiv:quant-ph/0412029, 2004

[13] J.-P. Chen et al., *"Sending-or-not-sending with independent lasers: Secure twin-field quantum key distribution over 509 km"*, Phys. Rev. Lett., vol. 124, no. 7, Feb. 2020, Art. no. 070501.

[14] M. Pittaluga et al., *"600-km repeater-like quantum communications with dual-band stabilization"*, Nat. Photon., vol. 15, no. 7, pp. 530-535, Jul. 2021.

[15] Feihu Xu, Xiongfeng Ma, Qiang Zhang, Hoi-Kwong Lo, Jian-Wei Pan, *"Secure quantum key distribution with realistic devices"*, arXiv:1903.09051 [quant-ph].

[16] Bennett, C.H., *"Quantum Cryptography Using Any Two Nonorthogonal States"*, Physical Review Letters, 68, 3121-3124. http://dx.doi.org/10.1103/PhysRevLett.68.3121

[17] A. K. Ekert, *"Quantum cryptography based on bell's theorem"*, Phys. Rev. Lett., vol. 67, August 1991, pp. 661-663, DOI 10.1103/PhysRevLett.67.661

[18] Xiaoliang Wu, Alexander Kolar, Joaquin Chung, Dong Jin, Tian Zhong, Rajkumar Kettimuthu, Martin Suchara, *"SeQUeNCe: A Customizable Discrete-Event Simulator of Quantum Networks"*, arXiv:2009.12000 [quant-ph]

[19] Le Boudec, Jean-Yves, *"Performance Evaluation of Computer and Communication Systems"*, https://leboudec.github.io/perfeval/

[20] Bruno Rijsman, *"The Cascade information reconciliation protocol."*, https://cascade-python.readthedocs.io/en/latest/protocol.html

[21] Mehic, M., Niemiec, M., Siljak, H., Voznak, M., *"Error Reconciliation in Quantum Key Distribution Protocols."*, https://doi.org/10.1007/978-3-030-47361-7_11

[22] *"NetworkX"*, https://networkx.org/

[23] Shannon, Claude E., *"Communication Theory of Secrecy Systems"*, Bell System Technical Journal. 28 (4): 656715. doi:10.1002/j.1538-7305.1949.tb00928.x. hdl:10338.dmlcz/119717. Retrieved 2011-12-21.

[24] Diffie, Whitfield; Hellman, Martin E., *"New Directions in Cryptography"*, IEEE Transactions on Information Theory. IT-22 (6): 646. Retrieved 8 December 2021.

[25] Brassard, G., Salvail, L. (1994)., *"Secret-Key Reconciliation by Public Discussion"*, In: Helleseth, T. (eds) Advances in Cryptology - EUROCRYPT '93. EUROCRYPT 1993. Lecture Notes in Computer Science, vol 765. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-48285-7_35

[26] Peppino Fazio, Mauro Tropea, Miralem Mehic, Floriano De Rango, Miroslav Voznak, *"Performance Evaluation of Free Space Optics Laser Communications for 5G and Beyond Secure Network Connections"*, https://www.scitepress.org/Papers/2023/120797/120797.pdf