THE STUDY OF HIGH-PERFORMANCE TECHNIQUES USING QUBITS IN QUANTUM CRYPTOGRAPHY

A PROJECT REPORT

Submitted by

ANANYA BANERJEE [Reg No: RA1511004010350]
INDUVALLI [Reg No: RA1511004010450]
AISHWARYA BALAJE [Reg No: RA1511004010463]
SONALI SHARMA [Reg No: RA1511004010474]

Under the guidance of

Mr.S PRAVEEN KUMAR

(Assistant Professor, Department of Electronics and Communication & Engineering)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

ELECTRONICS AND COMMUNICATION ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Kancheepuram District

APRIL 2019

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report "**THE STUDY OF HIGH-PERFORMANCE TECHNIQUES USING QUBITS IN QUANTUM CRYPTOGRAPHY**" is the Bonafide work of **Ms. Ananya Banerjee (Reg.No.RA1511004010350), Ms. Induvalli (Reg.No.RA1511004010450), Ms. Aishwarya Balaje (Reg.No.RA1511004010463) and Ms. Sonali Sharma (Reg.No.RA1511004010474)**, who carried out the project work under my supervision along with her batch mates. Certified further, that to the best of my knowledge, the work reported herein does not form any other project report on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

Date:	Project Supervisor (Dr. J. Subhashini)	Head of the Department (Dr. T. Rama Rao)
Date:	Examiner Internal	Examiner External

ABSTRACT

In the current age, data security has become one of the foremost priorities of the existing organizations, this is majorly due to the value that is attached to the information. Hence the existence of nefarious individuals whose sole purpose is to obtain such data is inevitable. This led researchers to look towards other means of encrypting as data, thus the fragile nature of quantum particles was considered appropriate for this purpose. Although the work done in the field of quantum cryptography is primarily theoretical, the purpose of this project is to showcase its implementation in an observable manner. Thus, this project establishes a more secure network using QKD for data transfer between sender and receiver and also enables the quick identification of an eavesdropper in the said network. Due to the complex nature of quantum networks, a physical implementation of the same is not feasible. Hence a simulation has been is implemented via the use of NS-3 (Network Simulator Version 3) which has QKDNetSim module built into it. The module sup- ports simulation of the QKD network in an overlay mode in order to facilitate the public as well as quantum channels to run simultaneously. This can be achieved by creating node containers which hold the nodes in sets according to a dedicated channel between those particular nodes. These containers can be used to separate the quantum channel from the public one. Network protocols have been tested and will be implemented according to the channel (congestion control, congestion avoidance). By testing the public channel implementation, the QKD channel implementation, and TCP Congestion control channel we will be able to achieve the QKD overlay channel for at least five nodes. An analysis of the quantum channel traffic at ideal state and also during network disruption has been carried out. This project will try to closely emulate real-life scenarios via increasing the complexity of the network as well as introducing a different topologies and network protocols.

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to the management of SRMIST and the Director, for providing us with the necessary resources to undertake this project. I would like to thank the HOD of the department of ECE and all the professors for making the process of doing this project easier for us. I would also like to thank our guide Mr. S. Praveen Kumar, Assistant Professor, Dept. of ECE, for his valuable guidance, constant encouragement, timely help and for providing us an opportunity to do research in the field. All through the work, despite his busy schedule, he has extended cordial support to us for completing this research work.

I sincerely thank Dr. J. Subhashini for providing us the opportunity to embark on this project and their guidance and encouragement in carrying out this project work.

Ananya Banerjee (RA1511004010350)

Induvalli (RA1511004010450)

Aishwarya Balaje (RA1511004010463)

Sonali Sharma (RA1511004010474)

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

1. **QKD**: Quantum Key Distribution
2. **OTP**: One Time Pad
3. **NS-3**: Network Simulator 3
4. **QKDNetSim**: Quantum Key Distribution Network Simulator
5. **QPKD**: Quantum Public Key Distribution
6. **QBER**: Quantum Bit Error Rate
7. **RKE**: Raw Key Exchange
8. **TCP**: Transmission Control Protocol
9. **UDP**: User Datagram Protocol
10. **MTU**: Maximum Transmission Unit
11. **P2P**: Peer to Peer
12. **IPv4**: Internet Protocol version 4
13. **IPv6**: Internet Protocol version 6
14. **OSPF**: Open Shortest Path First
15. **DSDV**: Destination-Sequenced Distance-Vector Routing

CHAPTER 1

INTRODUCTION

In the current age, data security has become one of the foremost priorities of existing organizations, this is majorly due to the value that is attached to information. Hence the existence of nefarious individuals whose sole purpose is to obtain such data is in-evitable. This led researchers to look towards other means of encrypting as data, thus the fragile nature of quantum particles was considered appropriate for this purpose. Quantum communication takes advantage of the laws of quantum physics to protect data. These laws allow particles (i.e. Photons) to transmit data along optical cables. These particles are known as quantum bits, or qubits. Their super-fragile quantum state collapses to either 1 or 0, if hacker tries to observe them in transit. This enables the quick identification of an eavesdropper in the network.

1.1 What is QKD?

A cryptographic solution provided by Quantum cryptography is not perishable, instead it fortifies prime secrecy that is applied to QKD. It is a well-known technology in which two entities can relay information securely. In classic cryptography, data in the form of bits is used to encrypt and decrypt information whereas in quantum cryptography, photons or quantum particles and photons polarization (which is its quantized property) is used. This is known as qubit, which is a unit for quantum cryptography. The transmissions are highly secure as they follow the laws of quantum mechanics.

Quantum Key distribution refers to a method in which a private key is shared be-tween two parties using the quantum channel, this is authenticated via the public channel. The key is used to encode and decode messages exchanged by both parties over the public/classical channel. Due to the fragile nature if quantum particles, the presence of an eavesdropper can be accurately identified if it intercepts the quantum channel. In this case, the generation of the key is terminated by the QKD protocol. QKD is one of the most widely known methods of quantum cryptography, it provides information-theoretically secure (ITS) solution to the key exchange problem.

1.2 Two different approaches

Quantum cryptographic devices typically employ individual photons of light and take advantage of either the Heisenberg uncertainty principle or quantum entanglement.

Uncertainty principle: Measurement of properties in quantum physics cannot be done in the same way as classical physics. In the quantum scale, some of the physical properties of certain pair of particles are complementary. This statement defines Heisenberg's uncertainty Principle. The primary property utilized QKD is photon polarization.

Entanglement: This describes a state in which two or more quantum particles physical properties are strongly correlated. This property can facilitate the research in long- distance quantum key distribution.

Aside from the dependence on the two fundamental laws qubit transmission also depends on quantum properties such as superposition, which means they can represent multiple combinations of 1 and 0 simultaneously as well as decoherence, which causes them to decay and ultimately disappear while interacting with the environment hence making only point to point communication between two nodes feasible.

1.3 BB84 protocol: Charles H. Bennett and Gilles Brassard (1984)

Charles H. Bennett and Gilles Brassard in the year 1984 presented the first QKD protocol. This protocol is called the BB84, it utilizes the state polarizations quantum bits. It has two bases of measurements and four photon polarization states.

The BB84 protocol has gained recognition for not only being a pioneer in field of QKD, but also in being the one of the most used protocols in terms ease of execution.

Although few researches have proven that the BB84 protocol not entirely secure, it is still theoretically used by several QKD protocols.

The Uncertainty principle based QKD protocol BB84 is applied in this paper in order to visualize the process of data transfer through a quantum channel. We choose this protocol due to its prompt identification of the presence of an intruder in the quantum channel via this, and because its implementation in a simulation environment when compared to other protocols is easy.

The BB84 protocol begins with the transmission of photons which have four random quantum states, relating to two mutually conjugate bases, rectilinear and diagonal. The rectilinear basis has two polarizations namely 0 degree represented horizontally and 90 degree represented vertically. The diagonal basis has 45 degree and 135 degree. The measurement of these polarizations cannot be done simultaneously, as if done they randomize each other. Hence, if an intruder attempts to access information from the quantum channel this will change the polarization of the intercepted photon. Thus, alerting the users of the presence of an eavesdropper in the network.

CHAPTER 2

LITERATURE SURVEY

Secured data transmission has always been a matter of great interest and several encryption techniques for secured data transmission has been devised till now in this regard. However, with rapidly developing technology, the available encryption techniques are becoming more prone to attack every day. Focusing this issue, in this project, we propose a technique of encryption that will enhance the security level by a significant margin with the help of quantum cryptography. Quantum Key Distribution (QKD) is a next-generation security technology that exploits the properties of quantum mechanics to enable two parties to generate an unconditionally secure shared secretkey. Quantum key distribution (QKD) is the object of close attention and rapid progress since once first quantum computers are available classical cryptography systems will become partially or completely insecure. It is breaking new ground since unlike traditional cryptographic techniques its security is based on the laws of quantum mechanics. QKD are considered as complex systems as systems contain multiple interconnected electrical, optical, and electro-optical subsystems and computer-based controllers. Currently, there is no single simulation framework that supports a high-level system engineering analysis of QKD system architectures. Though the years the research in the field of quantum systems has grown more complex, thus the tools for assessing practical feasibility and implementation of theoretical concepts are the need of the hour [1]. Hence, we had to look into various software's and frameworks that could be suitable for modeling, simulation, and analysis of QKD systems [2].

All traditional cryptographic algorithms used in the network communication environments, relied on mathematical models and computational assumptions, are unsafe and apt by many attackers (quantum and man-in-the middle attacks). Quantum Key Distribution was invented as a means to more securely transfer information between two parties. Since 1984, several experimental attempts to design QKD protocols have been developed on the bases of quantum mechanical rules. These QKD protocols were represented by different algorithms with limited ability to stand up against quantum attacks. These are evaluated to determine the most functional QKD protocols in the cryptography field and explain every QKD protocol as well illustrates the features that were utilized in each protocol [3]. QKD becomes a significant trend of new cryptographic revolution. There are multiple quantum mechanical theories that are used for QKD implementation, certain papers provide a standard simulation for BB84 protocol and which describe the improvement of key generation and key distribution mechanisms. They, also validate the efficiency of the protocol presented using different security con- figurations. Finally, indicating that the QKD is susceptible to corporate with different security applications and achieve the key availability for such applications. However, it suffers from the higher rate of authentication cost [4].

With the rise of quantum cryptography, its elements, quantum key distribution protocols and quantum networks [5] many papers are focusing on the field of photonics to help with the transmission of data via fiber optics and to more profoundly understand the importance of polarization in QKD.

The potential threat to today's information security cannot be neglected, and efficient quantum computing algorithms already exist. There are multiple researches that overview QKD and the most conspicuous and prominent QKD protocols, their work- flow and security basement [6]. There are other methods of QKD implementation that have also been investigated but are beyond the scope of this project, one such method is using quantum entanglement. In this method an efficient and deterministic quantum key distribution protocol for establishing a secret key between two untrusted users is proposed. A secret key is distributed to a sender and a receiver who share entangled states with a third trusted party but not with each other. This secret key is distributed by the means of special pure quantum states using remote state preparation andcontrolled gates. In addition, parity bits of the entangled pairs are employed and the ancillary states to assist in preparing and measuring the secret states. Distributing a state to two users require two maximally entangled pairs as the quantum channel and a two-particle von Neumann projective measurement. This protocol is exact and deterministic. It distributes a secret key of d qubits by the means of 2d entangled pairs and, on average, d bits of classical communication. This research pertaining the demonstration of the degree of security of this protocol against entanglement attacks and present a method of privacy amplification has also been explored [7].

It is a known fact that the quantum bit exponentially decreases the computing time necessary to find a key. In addition, the no-cloning ensures eavesdroppers are not present in communication channels. Certain research suggests that quantum particles can not only ensure greater security bit also break existing cryptographic algorithms [8].

Another property of Quantum key distribution (QKD) is that it permits information-theoretically secure transmission of digital encryption keys, if the behavior of the devices employed for the key exchange can be reliably modeled and predicted. Remarkably, no assumptions must be made on the capabilities of an eavesdropper other than that she is bounded by the laws of nature, thus making the security of QKD unconditional. However, unconditional security is hard to achieve in practice. For example, any experimental realization can only collect finite data samples, leading to vulnerabilities against coherent attacks, the most general class of attacks, and for some protocols the theoretical proof of robustness against these attacks is still missing. For these reasons, in the past many QKD experiments have fallen short of implementing an unconditionally secure protocol and have instead considered limited attacking capabilities by the eavesdropper. Thus, the analysis of QKD network implementation under coherent at- tacks in the most challenging environment: the long-distance transmission of keys is an important area of study. It is demonstrated that the BB84 protocol can provide positive key rates for distances up to 240 km without multiplexing of conventional signals, and up to 200 km with multiplexing. Useful key rates can be achieved even for the longest distances, using practical thermo-electrically cooled single-photon detectors [9].

There are many advancements that have been made in the field of quantum cryptography even though they are purely hypothesis based, they depict the path in which this research is headed. The transmission of data in the quantum channel is done by means of fiber optics, hence the it is affected by the problems that exist in this form of data transmission. There is research that exists that counters such problems by referencing frame independent QKD protocol for polarization qubits and hence showing that it overcomes detrimental effects of drifting fiber birefringence in a polarization maintaining fiber [10]. The principle of quantum key distribution based on BB84 protocol and the realization of qubits extraction for free space quantum cryptography system [11] is a new area of study that is being looked into. Some research also suggests that all-graphene solid-state components can be used within quantum networks, due to their possible implementation as quantum repeaters [12]. There is also the possibility of implementing satellite to ground quantum key distribution based on the properties of entanglement [13]. A theoretical approach in the implementation of QKD in IoT to increase the security of wireless sensor-based networks as well as cellular networks [14].

Since practical implementation of QKD networks is not feasible thus multiple simulation methods are being adopted presently. In such simulation environments, several routing protocols in terms of the number of sent routing packets, throughput, bit error rate, QKD buffer concentration and Packet Delivery Ratio of data traffic flow using NS-3 simulator have been analyzed by certain researchers [15].

Unlike conventional networks, there are few software applications dealing with QKD. The Quantum Cryptography Protocol Simulator developed using C/C++ architecture can analyze the quantum bit error rate (QBER) and eavesdropper influence on the performances of the quantum channel when BB84 or B92 QKD protocol is used [16].

A simulation framework for the QKD protocols using Opti System is also in use. Despite the exitance of multiple methods to simulate QKD networks NS-3 was chosen since it is an opensource software. To the best of our knowledge, the applications for simulating the QKD networks with multiple nodes and links are not available, this due to the principle of decoherence. In practice, the quantum and the public channel can be implemented on the same media using existing optical components.

CHAPTER 3

RESEARCH METHODOLOGY

Presently, data security has become one of the foremost priorities of the existing organizations, this is majorly due to the value that is attached to information. Hence the existence of nefarious individuals whose sole purpose is obtain such data is inevitable. This led researchers to look towards other means of encrypting as data, thus the fragile nature of quantum particles was considered appropriate for this purpose. Although the work done in the field of quantum cryptography is primarily theoretical, the purpose of this project is to showcase its implementation in an observable manner.

This project establishes a more secure communications network using QKD for data transfer between sender and receiver and also enables the quick identification of an eavesdropper in the said network. Due to the complex as nature of quantum networks, a physical implementation of the same is not feasible. Hence a simulation of the same is implemented via the use of NS-3 (Network Simulator 3) with QKDNetSim built into it. The module supports simulation of the QKD network in an overlay mode in order to facilitate the public as well as quantum channels to run simultaneously.

Since, there is abundant theoretical research available in the field of Quantum Cryptography existing we decided to explore the following areas:

1. Implementation of observable networks in the Quantum Channel
2. Studying the factors that affect the transmission of data transfer in said channel
3. Establishing a link between network disruption and the presence of an eavesdropper
4. Implementation of networks in overlay form

In order to arrive at the above objectives, we decided to go about our research in stages. First, we tried to gain an understanding of what exactly is QKD and why is it considered so secure. For this we had to study the various laws within Quantum Mechanics that are employed in this field. Second, we tried to identify the various protocols that can be implemented within Quantum Key Distribution. Based on the available data we arrived at the conclusion that BB84 was the most easily executable and one of the most widely used protocols in QKD implementation. Then, we tried to narrow down the type of software that is capable of implementing the required simulation with being open source as well. Hence, we choose NS-3, in order to implement quantum channel networks with it we had to build QKDNetSim into it. We came across QKDNetSim by going through the research done by Miralem Mehic´ on the topic of "Using Quantum Key Distribution for Securing Real-Time Applications". His papers proved to be extremely beneficial for our research.

Even though we have formulated a working QKD network simulation, our project has several limitations which can only be addressed with further research in this area:

1. The quantum channel can only implement point to point networks.
2. The system cannot simulate overly complicated networks within the quantum channel due to network traffic. There are no congestion control features built into the quantum channel networks unlike the public channel.
3. Since the base theory of this network is the Uncertainty Principle, it does not support long distance networks at the moment. This can be modified in the future.
4. The implementation is primarily simulation based; hence it cannot emulate all the physical constrains that the channel could undergo in real-life.

The chaptalization that has been implemented in this report contains five sections. First, deals with an introduction to our area of study. Second, focusses on the various papers that we went through in order to develop this project. Third, deals with the research methodology. Fourth, describes the system model and interface. Fifth, deals with the results and the conclusion.

CHAPTER 4

SYSTEM MODEL

The system that has been implemented in this project woks on the basis of the Uncertainty Principle, and the simulation consists of a public and a quantum channel both of which have been implemented on NS-3. This section contains an overview of the theoretical model as well as the simulation procedure.

4.1 Theoretical Representation

This model deals with the method of exchange of information in a secured manner using quantum key distribution which is free from eavesdropping and is confidential.

Although Quantum key distribution as a complicated field of study, its functioning can be summarized in three stages:

1. Key Exchange: The raw key is generated and is exchanged in the form of Qubits between the two parties.
2. Key Sifting: In this stage only certain cases from the raw key are selected. After the sifting step, both parties share a sequence of bits, called the sifted key.
3. Key Distillation: The shifted key is jointly processed by the sender and the receiver jointly to extract the secure sequence of bits called secret key. It consists of three steps; error correction, privacy amplification, authentication.

QKD protocols define only the first two steps mentioned above.

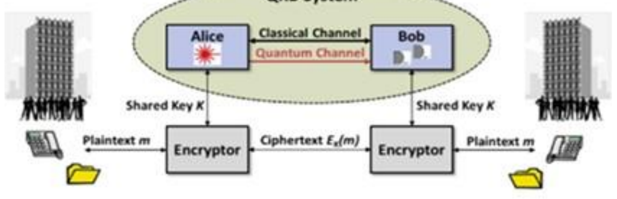


Figure 4.1: QKD System Schematic

In QPKD, the communicating parties use two communication channels namely a classical/public channel and a quantum channel. They transmit polarized single photons i.e. qubits on the quantum channel and the conventional messages on classical channel. The following are the steps for secret key which is shared between two users.

1. The sender creates a sequence of random bits and chooses random bases. The bits are represented as photons and are sent through the quantum channel.
2. At the receiver end, each bit is measured by passing it through one of the two basis.
3. The sender and the receiver will share the same key only if they choose the same base sequence.
4. The receiver relays his/her chosen bases to the sender by the public channel and the sender replies with a message stating which of them are correct.
5. Both the parties will delete the bits which are of different bases and the other bits are the key known as sifted key.

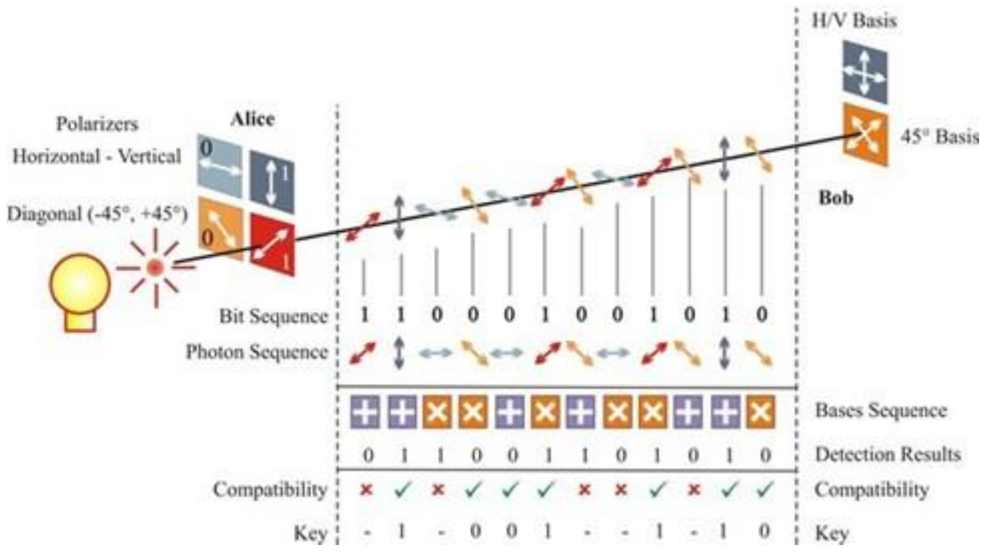

H/V Basis
45° Basis
Bob
Polarizers
Horizontal - Vertical
Alice
Diagonal (-45°, +45°)
Bit Sequence 1 1 0 0 0 1 0 0 1 0 1 0
Photon Sequence
Bases Sequence
Detection Results 0 1 1 0 0 1 1 0 1 0 1 0
Compatibility ✗ ✓ ✗ ✓ ✓ ✓ ✗ ✗ ✓ ✗ ✓ ✓
Key - 1 - 0 0 1 - - 1 - 1 0

Figure 4.2: Key exchange in the BB84 protocol implemented with polarization of photons

4.2 Software Specification

The process of setting up the software in order to implement the simulations for QKD has been done with the help of qkdnetsim-dev github repository as well as ns3 wiki pages. The basic hardware requirement for this software to run is a computer with 2GB RAM dedicated to the Linux virtual environment. However, in this project the virtual environment had been assigned 4GB of RAM in order to ensure a smooth simulation process. NS-3 is already a well-known network simulation tool since it has a variety of libraries built into it. QKDNetSim is used for the quantum channel simulation since it has various cryptographic algorithms built into it. Also, it satisfies certain requirements:

1. Accuracy: It is able to closely emulate real world variable that hinder
2. communications across the quantum channel.
3. Extensibility: It is reflected in terms of the ease of upgrade, that is, implementation of the new models and solutions in the field of QKD and other technologies.
4. Usability: It reflects its ease of use and integration with the existing software.
5. Availability: It deals with its availability when compared to other resources. QKD-NetSim and NS-3 are opensource software’s.

Multiple software’s have been used to implement the network simulation. These are as follows:

1. Virtual Machine: VMware 15 Player
2. Operating System: Ubuntu Desktop 18.04.1 LTS (Linux OS)
3. Network Simulator: ns-3.29
4. Libraries built into ns-3: gcc, g++, python, mercurial etc as mentioned in www.nsnam.org/wiki.
5. Network animator: NetAnim
6. Graphical interface: Gnuplot
7. QKD Simulator: QKDNetSim
8. Integrated Development Environment: Eclipse, Turbo 3.1, IDLE 3.6

4.3 Network Model Implementation

The process used for implementing the simulation of the networks in public as well as quantum channel in NS-3 follows the steps shown in the flowchart. These steps are common for implementation of any network on this platform.

1. Topology Definition: since, it’s possible to implement these networks in public as well as the quantum channel, it is also possible to integrate the nodes of the network to form various arrangements (like bus, ring, star, mesh) with the help of links.

2. Model Development: this speaks of the various protocols (TCP, UDP) that can be implemented in the network in the NS-3 by programming it along with the nodal orientation
3. Node and Link Configuration: these are additional node and link specifications (Baud rate, Bit rate, public channel congestion control, etc.).
4. Execution: when the implemented network is simulated in the NS-3, the results are two folds (ASCII and graphical).
5. Performance Analysis: this is the ASCII result that tell the client-server relation, the total number of packets sent, the total data transmitted, the client and server address along with the transmitting and receiving ports. In the case of public channel, it also tells the number of packets dropped and at what time.
6. Graphical Visualization: by using two software two different graphical visualization of the network is achieved. One, is a network animator that simulates the data flow for the total time duration. The other is a plotting aid useful to determine the total data transmitted (in bits), for what period (s), and BER

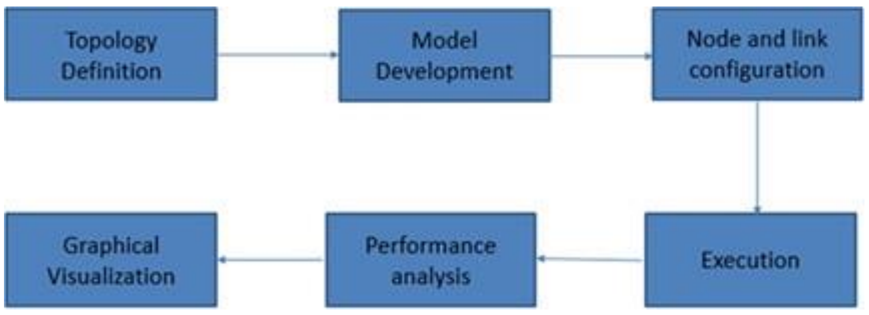


Figure 4.3: Network model implementation flowchart

4.4 Quantum Channel Variables

Due to the nature of the single photon propagation, QKD networks are mainly limited to the metropolitan scale in which the network can be geographically divided into multiple domains (autonomous systems), or in the simplest case it can be a simple network in a single domain.

The QKDNetSim module that has been built into NS-3 enabled us to simulate and analyze characteristics of the quantum channel. This contains the following features:

1. QKD Key: This describes the key being used for encryption.
2. QKD Buffer: The keys are stored in the buffer. The analysis of the buffer concentration describes the traffic in the channel. More about this is discussed in a latersection.
3. QKD Crypto: This class of the module used to perform encryption, decryption, authentication and reassembly of previously
4. fragmented packets.
5. QKD Virtual Network Device: It facilitates the operation of the overlay routing protocol.
6. QKD Post-processing Application: It deals with the extraction of the secret key from the raw key transmitted over the quantum channel.
7. QKD Graph: This class of the module enables the easy extraction of the graphs related to the QKD buffer states.

The element of this module that this paper is primarily focused on is the QKD Buffer. This has many variables that further facilitate the process of analyzing the quantum channel. Endpoints of links which contain the buffer are gradually filled with the new key material and subsequently used for the encryption/decryption of data flow [8]. The key consumption rate depends on the encryption algorithm used and the network traffic, while the key rate of the link determines the key charging rate. If there isn't enough key material in the storage, encryption of data flow cannot be performed [9] and QKD link can be characterized as "currently unavailable". Key material storage has a limited capacity and QKD devices constantly generate keys at their maximum rate until key storages are filled. Hence any disruption in the link can also compromise the key rates, these changes can be observed in the QKD buffer graphs. The variables used to define said graphs are, *Mcur* - current buffer capacity, *Mmin* – minimum pre-shared key material, *Mmax* – maximum storage depth, *Mcur* – current key concentration in the buffer, *Mthr* – threshold value.

The QKD buffer can be in one of the following states:

1. READY: $Mcur \geq Mthr$,
2. WARNING:$Mthr > Mcur$ (t) $> Mmin$, the previous state was READY,
3. CHARGING: $Mthr > Mcur$ (t), the previous state was EMPTY,
4. EMTPY: $Mmin \geq Mcur$ (t), the previous state was WARNING or CHARGING.

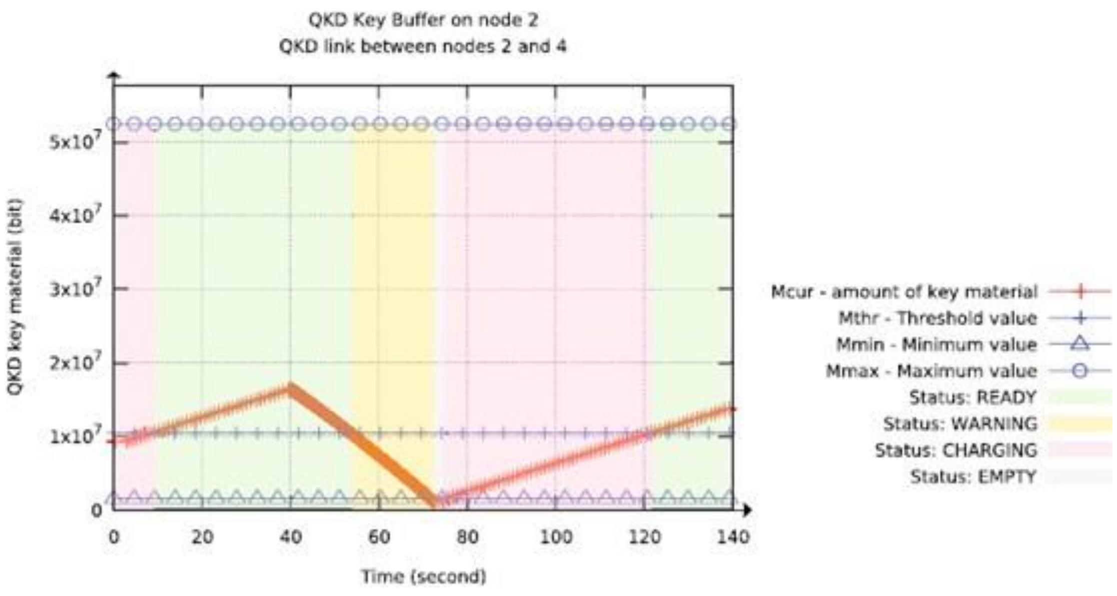
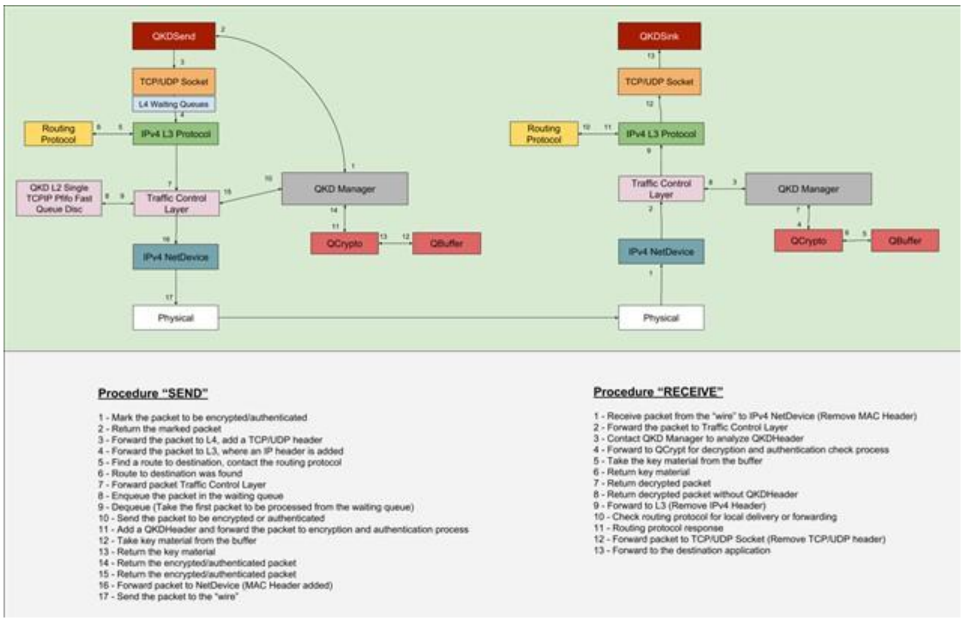


Figure 4.4: Graphical representation of QKD buffer

4.5 QKD network module organization



Procedure "SEND"

1 - Mark the packet to be encrypted/authenticated
2 - Return the marked packet
3 - Forward the packet to L4, add a TCP/UDP header
4 - Forward the packet to L3, where an IP header is added
5 - Find a route to destination, contact the routing protocol
6 - Route to destination was found
7 - Forward packet Traffic Control Layer
8 - Enqueue the packet in the waiting queue
9 - Dequeue (Take the first packet to be processed from the waiting queue)
10 - Send the packet to be encrypted or authenticated
11 - Add a QKDHeader and forward the packet to encryption and authentication process
12 - Take key material from the buffer
13 - Return the key material
14 - Return the encrypted/authenticated packet
15 - Return the encrypted/authenticated packet
16 - Forward packet to NetDevice (MAC Header added)
17 - Send the packet to the "wire"

Procedure "RECEIVE"

1 - Receive packet from the "wire" to IPv4 NetDevice (Remove MAC Header)
2 - Forward the packet to Traffic Control Layer
3 - Contact QKD Manager to analyze QKDHeader
4 - Forward to QCrypt for decryption and authentication check process
5 - Take the key material from the buffer
6 - Return key material
7 - Return decrypted packet
8 - Return decrypted packet without QKDHeader
9 - Forward to L3 (Remove IPv4 Header)
10 - Check routing protocol for local delivery or forwarding
11 - Routing protocol response
12 - Forward packet to TCP/UDP Socket (Remove TCP/UDP header)
13 - Forward to the destination application

Figure 4.5: QKDNetSim module organization with a single TCP/IP stack

4.6 Result and Interpretation

The implementation of this project has been done in two parts, firstly we have tried to implement a normal public channel at ideal conditions as well as a public channel network with congestion control. Secondly, we have tried to run a simple quantum channel network at ideal conditions, after which we have tried to implement a more complicated overlay network of the same to identify the changes in the channel traffic. The ideal values of *Mmin*, *Mmax*, *Mcur* can be set in the program used to design the network. The Mthr depends on the network topology. It can be calculated using specific formulae [10] The threshold value *Mthr* is proposed to increase the stability of QKD links, where it holds that *Mthr*≤ *Mmax*.

• Each node a calculates value La summarizing the Mcur values of links to its neighbors j and dividing it with the number of its neighbors Na, that is:

$$L_a = \frac{\sum_j^N M_{cur,a,j}}{N_a}, \forall j \in N_a$$

• Then, each node exchanges calculated value La with its neighbors. The minimum value is accepted as the threshold value of the link, that is:

$$M_{thr,a,b} = min\{L_a, L_b\}$$

By using Mthr, the node gains information about the statuses of network links. The higher the value, the better the state of links that are more than one hop away. Since the protocol used by the networks in both channels is Open Shortest Path First (OSPF), a higher threshold value can be chosen. Depending on this, paths can be rerouted or terminated. Hence, the QKD buffer capacity graph enables us to analyze the traffic in the current path, and choose a more efficient path for data exchange. A dip in the QKD buffer capacity indicates that parts of the key have been dropped. This can be due to congestion or the presence on an intruder in the path. Path congestion in the quantum channel is unlikely since the connection between the nodes is point to point. This is because the key material is primarily in the form of qubits and it follows quantum properties. The addition of constrains due to the environment need not be implemented in the simulation, as data transfer via the quantum channel is done using fiber optic cables.

4.6.1 Test Cases in public channel

1. Public channel implementation (first.cc)

This example displays the implementation of a simple network that relays information in the public channel.

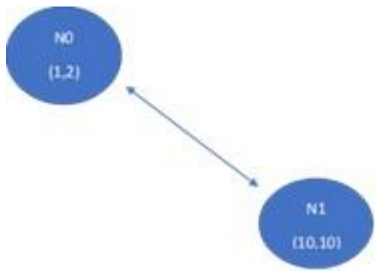
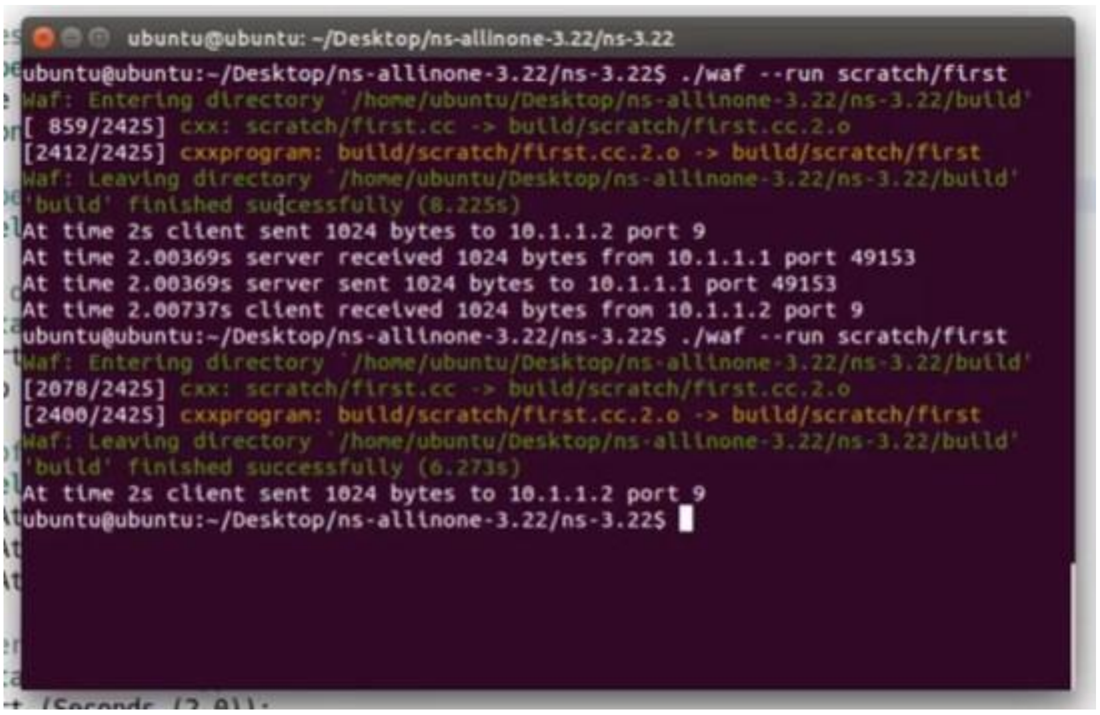


Figure 4.6: first.cc network



```
ubuntu@ubuntu:~/Desktop/ns-allinone-3.22/ns-3.22$ ./waf --run scratch/first
Waf: Entering directory `/home/ubuntu/Desktop/ns-allinone-3.22/ns-3.22/build'
[ 859/2425] cxx: scratch/first.cc -> build/scratch/first.cc.2.o
[2412/2425] cxxprogram: build/scratch/first.cc.2.o -> build/scratch/first
Waf: Leaving directory `/home/ubuntu/Desktop/ns-allinone-3.22/ns-3.22/build'
'build' finished successfully (8.225s)
At time 2s client sent 1024 bytes to 10.1.1.2 port 9
At time 2.00369s server received 1024 bytes from 10.1.1.1 port 49153
At time 2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
At time 2.00737s client received 1024 bytes from 10.1.1.2 port 9
ubuntu@ubuntu:~/Desktop/ns-allinone-3.22/ns-3.22$ ./waf --run scratch/first
Waf: Entering directory `/home/ubuntu/Desktop/ns-allinone-3.22/ns-3.22/build'
[2078/2425] cxx: scratch/first.cc -> build/scratch/first.cc.2.o
[2400/2425] cxxprogram: build/scratch/first.cc.2.o -> build/scratch/first
Waf: Leaving directory `/home/ubuntu/Desktop/ns-allinone-3.22/ns-3.22/build'
'build' finished successfully (6.273s)
At time 2s client sent 1024 bytes to 10.1.1.2 port 9
ubuntu@ubuntu:~/Desktop/ns-allinone-3.22/ns-3.22$ █
```

Figure 4.7: Output for first.cc

2. Public channel implementation with graphical output (seventh.cc)

This example displays the implementation of a more complex network with congestion control capabilities that relays information in the public channel.

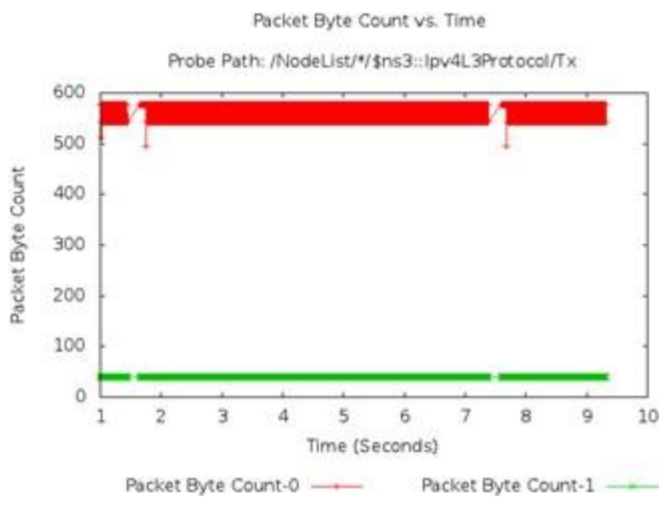


Figure 4.8: Packet count vs time graph for seventh.cc

4.6.2 Test cases in Quantum channel

1. Quantum channel implementation (qkdchanneltest.cc)

This example displays the implementation of a simple network that relays information in the quantum channel. It has an ideal QKD buffer graph as it doesn't loose any of the data it transmits.

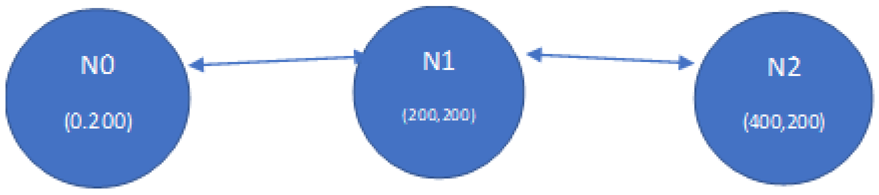
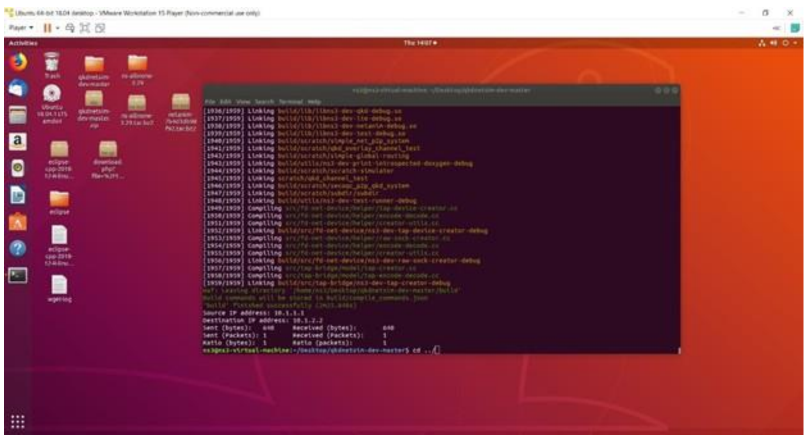


Figure 4.9: qkdchanneltest.cc network



```
[1936/1959] Linking build/lib/libns3-dev-qkd-debug.so
[1937/1959] Linking build/lib/libns3-dev-lte-debug.so
[1938/1959] Linking build/lib/libns3-dev-netanim-debug.so
[1939/1959] Linking build/lib/libns3-dev-test-debug.so
[1940/1959] Linking build/scratch/simple_net_p2p_system
[1941/1959] Linking build/scratch/qkd_overlay_channel_test
[1942/1959] Linking build/scratch/simple-global-routing
[1943/1959] Linking build/utils/ns3-dev-print-introspected-doxygen-debug
[1944/1959] Linking build/scratch/scratch-simulator
[1945/1959] Linking scratch/qkd_channel_test
[1946/1959] Linking build/scratch/secoqc_p2p_qkd_system
[1947/1959] Linking build/scratch/subdir/subdir
[1948/1959] Linking build/utils/ns3-dev-test-runner-debug
[1949/1959] Compiling src/fd-net-device/helper/tap-device-creator.cc
[1950/1959] Compiling src/fd-net-device/helper/encode-decode.cc
[1951/1959] Compiling src/fd-net-device/helper/creator-utils.cc
[1952/1959] Linking build/src/fd-net-device/ns3-dev-tap-device-creator-debug
[1953/1959] Compiling src/fd-net-device/helper/raw-sock-creator.cc
[1954/1959] Compiling src/fd-net-device/helper/encode-decode.cc
[1955/1959] Compiling src/fd-net-device/helper/creator-utils.cc
[1956/1959] Linking build/src/fd-net-device/ns3-dev-raw-sock-creator-debug
[1957/1959] Compiling src/tap-bridge/model/tap-creator.cc
[1958/1959] Compiling src/tap-bridge/model/tap-encode-decode.cc
[1959/1959] Linking build/src/tap-bridge/ns3-dev-tap-creator-debug
Waf: Leaving directory `/home/ns3/Desktop/qkdnetsim-dev-master/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully ([illegible])
Source IP address: 10.1.1.1
Destination IP address: 10.1.2.2
Sent (bytes):   640     Received (bytes):       640
Sent (Packets): 1       Received (Packets):     1
Ratio (bytes):  1       Ratio (packets):        1
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master$ cd ../
```

Figure 4.10: Output for qkdchanneltest.cc

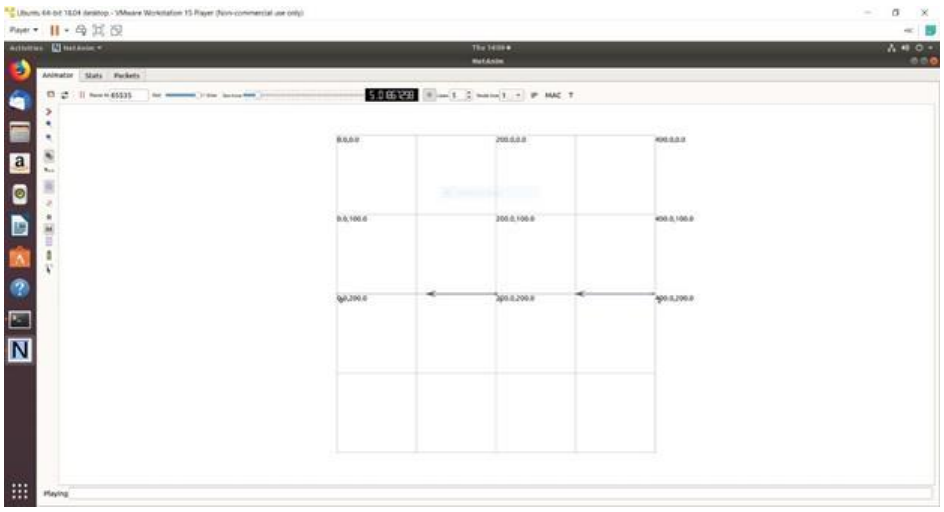


Figure 4.11: qkdchanneltest.cc network animation

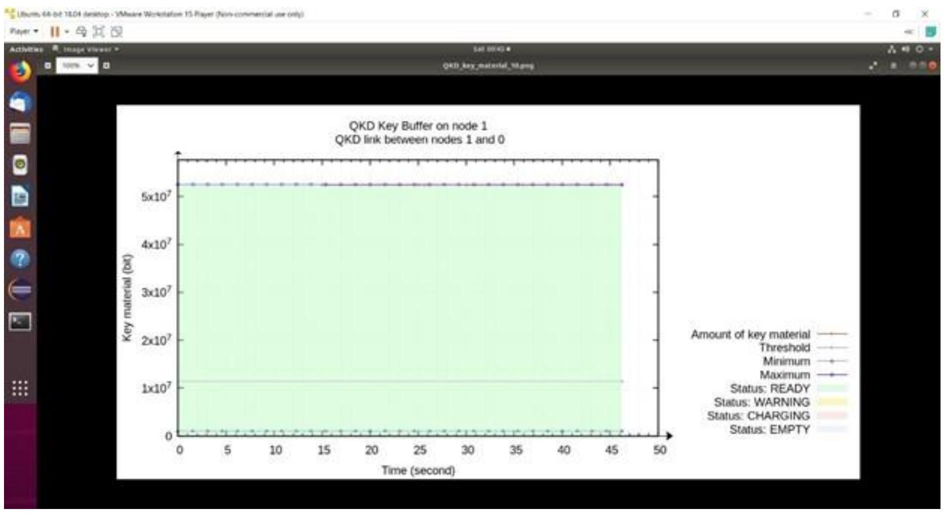


Figure 4.12: QKD buffer capacity between nodes 1 and 0

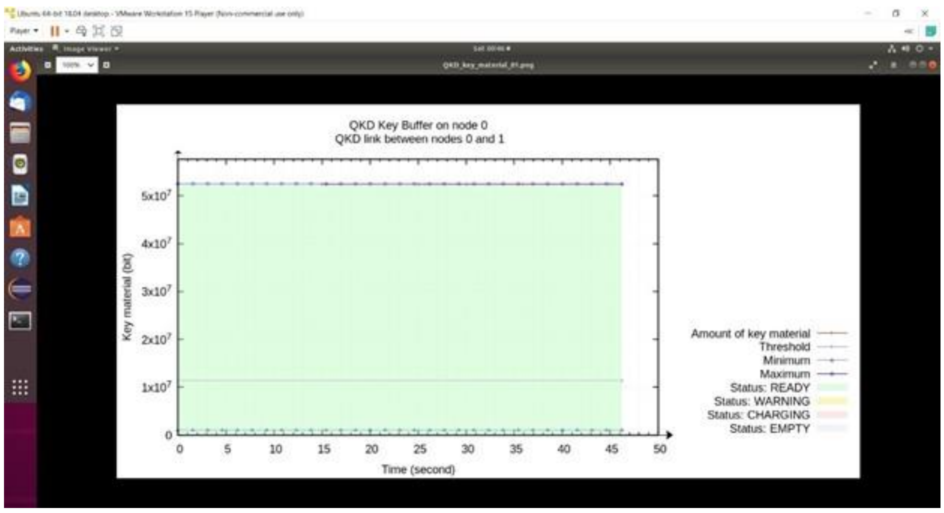


Figure 4.13: QKD buffer capacity between nodes 0 and 1

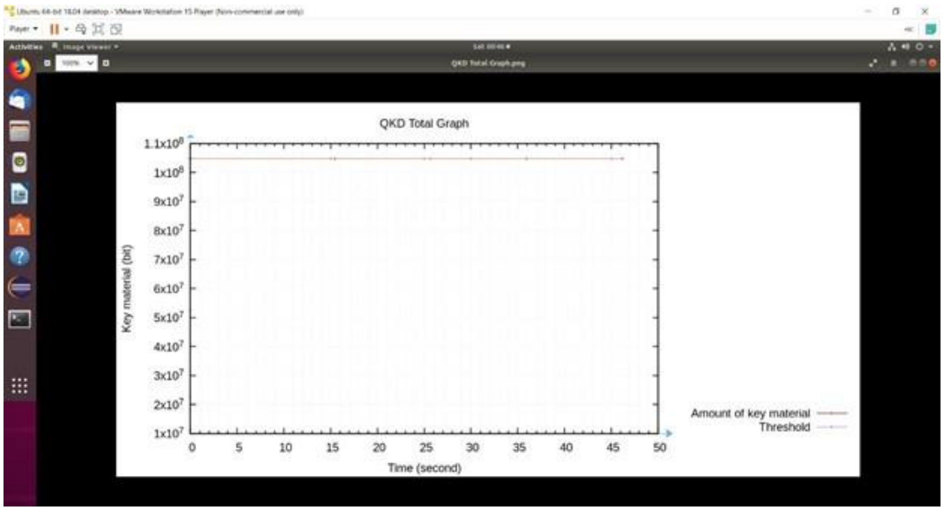
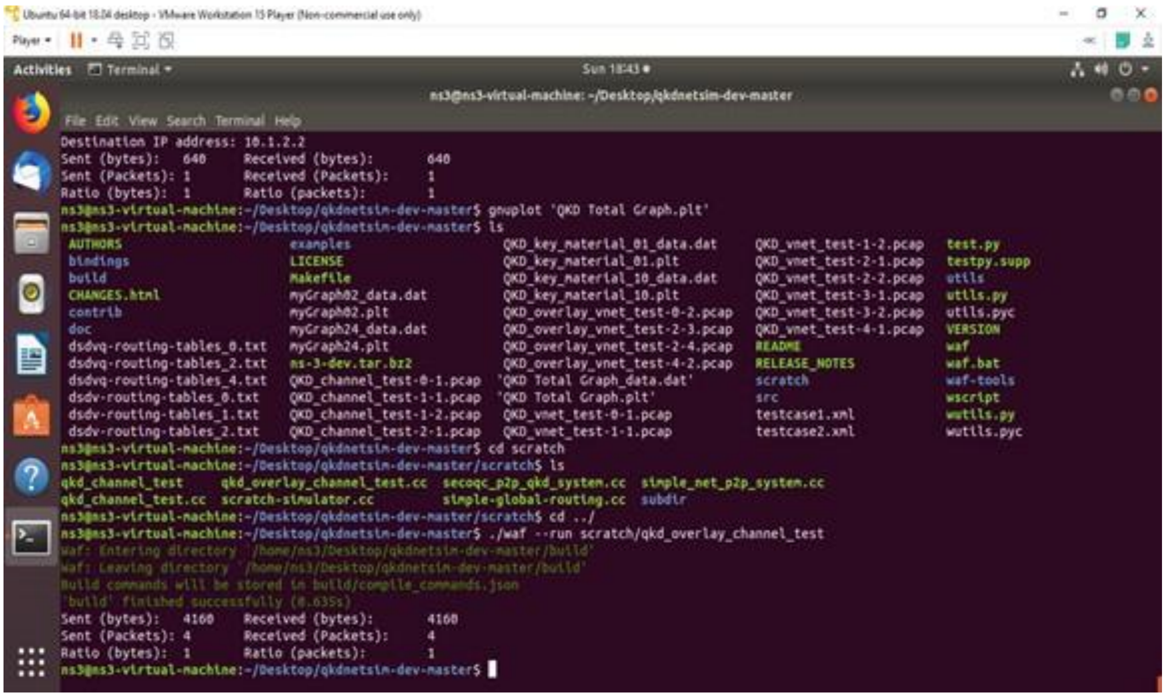


Figure 4.14: Total QKD buffer capacity

2. Quantum channel implementation for overlay network (qkdoverlaychanneltest.cc) This example displays the implementation of a network with more nodes and an overlay between the public and quantum channel. The graphs show the amount of data sent and received as well as the QKD buffer concentration.



```
ns3@ns3-virtual-machine: ~/Desktop/qkdnetsim-dev-master
Destination IP address: 10.1.2.2
Sent (bytes):   640     Received (bytes):       640
Sent (Packets): 1       Received (Packets):     1
Ratio (bytes):  1       Ratio (packets):        1
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master$ gnuplot 'QKD Total Graph.plt'
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master$ ls
 AUTHORS                    examples                  QKD_key_material_01_data.dat     QKD_vnet_test-1-2.pcap   test.py
 bindings                   LICENSE                   QKD_key_material_01.plt          QKD_vnet_test-2-1.pcap   testpy.supp
 build                      Makefile                  QKD_key_material_10_data.dat     QKD_vnet_test-2-2.pcap   utils
 CHANGES.html               myGraph02_data.dat        QKD_key_material_10.plt          QKD_vnet_test-3-1.pcap   utils.py
 contrib                    myGraph02.plt             QKD_overlay_vnet_test-0-2.pcap   QKD_vnet_test-3-2.pcap   utils.pyc
 doc                        myGraph24_data.dat        QKD_overlay_vnet_test-2-3.pcap   QKD_vnet_test-4-1.pcap   VERSION
 dsdvq-routing-tables_0.txt myGraph24.plt             QKD_overlay_vnet_test-2-4.pcap   README                   waf
 dsdvq-routing-tables_2.txt ns-3-dev.tar.bz2          QKD_overlay_vnet_test-4-2.pcap   RELEASE_NOTES            waf.bat
 dsdvq-routing-tables_4.txt QKD_channel_test-0-1.pcap 'QKD Total Graph_data.dat'       scratch                  waf-tools
 dsdv-routing-tables_0.txt  QKD_channel_test-1-1.pcap 'QKD Total Graph.plt'            src                      wscript
 dsdv-routing-tables_1.txt  QKD_channel_test-1-2.pcap QKD_vnet_test-0-1.pcap           testcase1.xml            wutils.py
 dsdv-routing-tables_2.txt  QKD_channel_test-2-1.pcap QKD_vnet_test-1-1.pcap           testcase2.xml            wutils.pyc
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master$ cd scratch
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master/scratch$ ls
qkd_channel_test     qkd_overlay_channel_test.cc  secoqc_p2p_qkd_system.cc  simple_net_p2p_system.cc
qkd_channel_test.cc  scratch-simulator.cc         simple-global-routing.cc  subdir
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master/scratch$ cd ../
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master$ ./waf --run scratch/qkd_overlay_channel_test
Waf: Entering directory `/home/ns3/Desktop/qkdnetsim-dev-master/build'
Waf: Leaving directory `/home/ns3/Desktop/qkdnetsim-dev-master/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (0.635s)
Sent (bytes):   4160    Received (bytes):       4160
Sent (Packets): 4       Received (Packets):     4
Ratio (bytes):  1       Ratio (packets):        1
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master$
```

Figure 4.15: Output for qkdoverlaychanneltest.cc

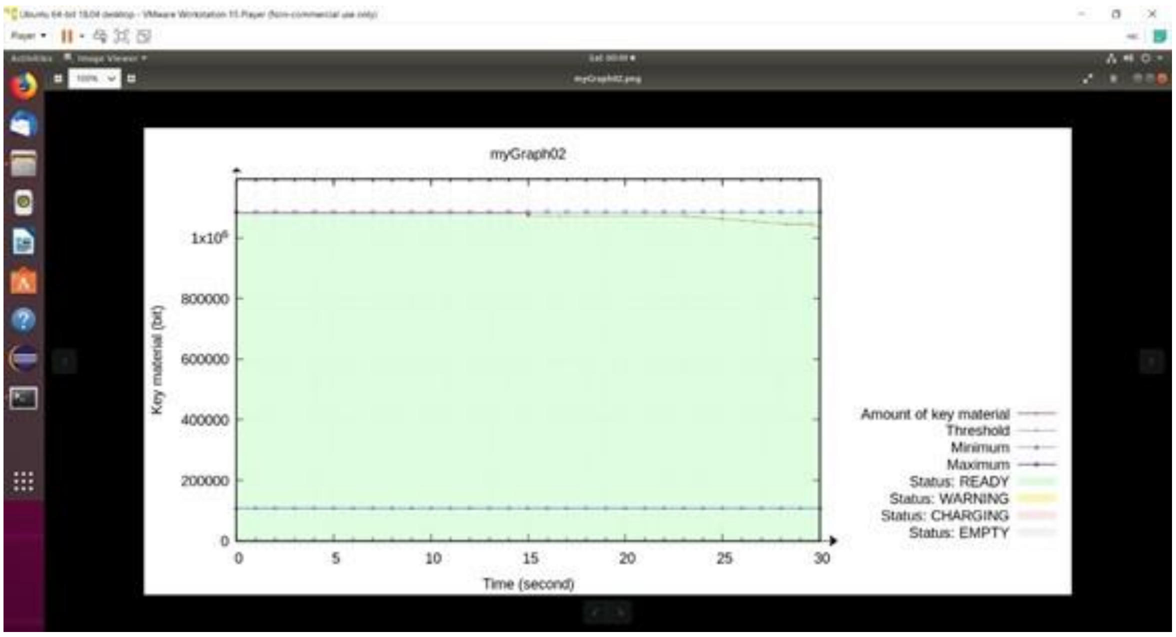


Figure 4.16: QKD buffer capacity between nodes 0 and 2

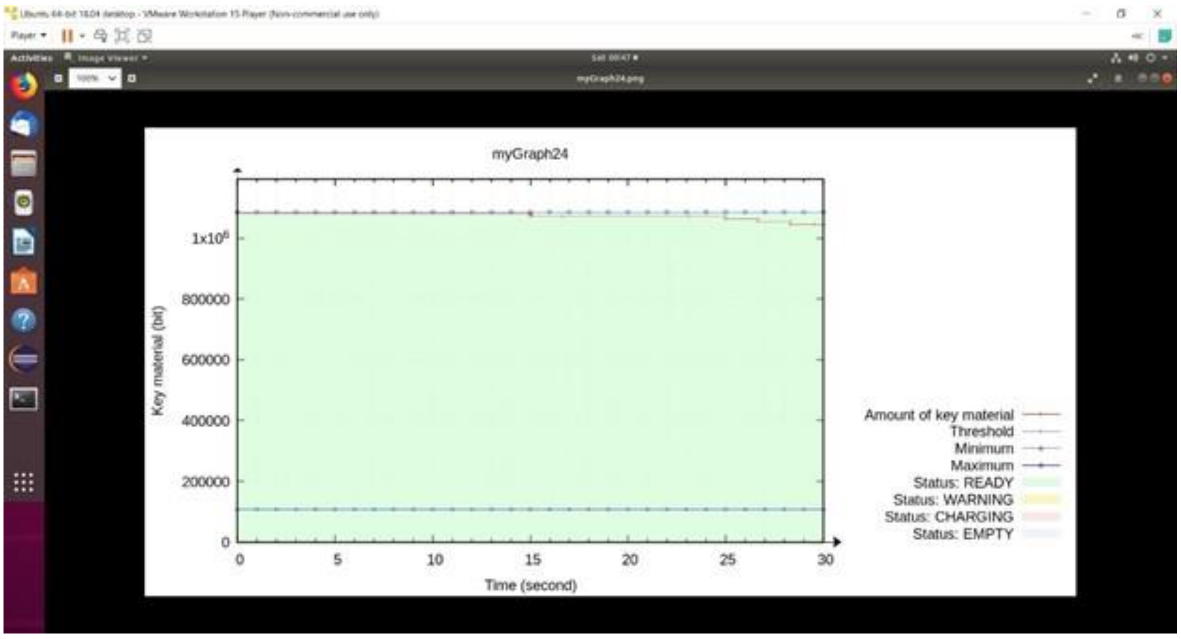
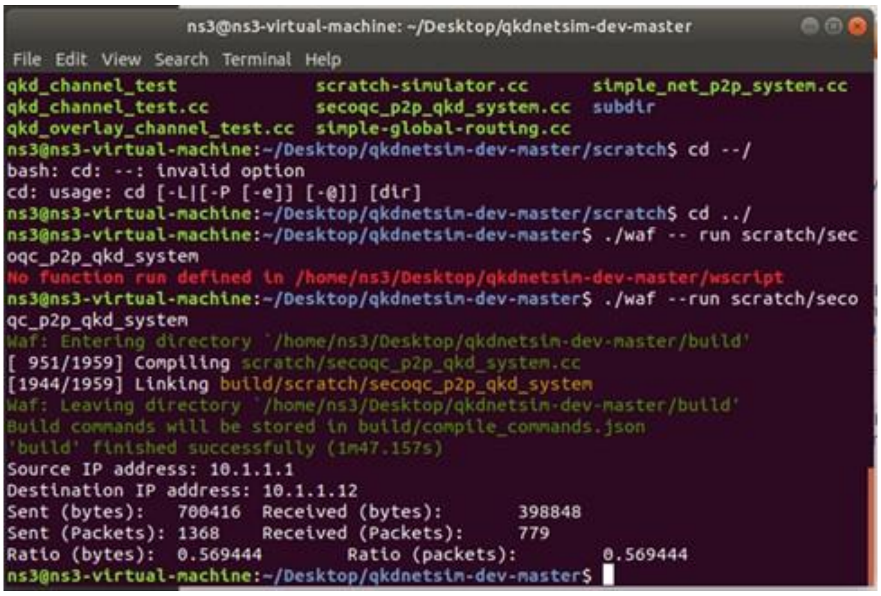


Figure 4.17: QKD buffer capacity between nodes 2 and 4

3. Quantum channel implementation for peer to peer network (secoqcp2pqkdsystem.cc) This example displays the implementation of a complex network that relays information in the quantum channel. The QKD buffer graphs are not ideal due to channel disruption.


ns3@ns3-virtual-machine: ~/Desktop/qkdnetsim-dev-master
File Edit View Search Terminal Help

```
qkd_channel_test              scratch-simulator.cc      simple_net_p2p_system.cc
qkd_channel_test.cc           secoqc_p2p_qkd_system.cc  subdir
qkd_overlay_channel_test.cc   simple-global-routing.cc
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master/scratch$ cd --/
bash: cd: --: invalid option
cd: usage: cd [-L|[-P [-e]] [-@]] [dir]
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master/scratch$ cd ../
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master$ ./waf -- run scratch/sec
oqc_p2p_qkd_system
No function run defined in /home/ns3/Desktop/qkdnetsim-dev-master/wscript
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master$ ./waf --run scratch/seco
qc_p2p_qkd_system
Waf: Entering directory `/home/ns3/Desktop/qkdnetsim-dev-master/build'
[ 951/1959] Compiling scratch/secoqc_p2p_qkd_system.cc
[1944/1959] Linking build/scratch/secoqc_p2p_qkd_system
Waf: Leaving directory `/home/ns3/Desktop/qkdnetsim-dev-master/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (1m47.157s)
Source IP address: 10.1.1.1
Destination IP address: 10.1.1.12
Sent (bytes):   700416  Received (bytes):       398848
Sent (Packets): 1368    Received (Packets):     779
Ratio (bytes):  0.569444        Ratio (packets):        0.569444
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master$
```

Figure 4.18: Output for secoqcp2pqkdsystem.cc

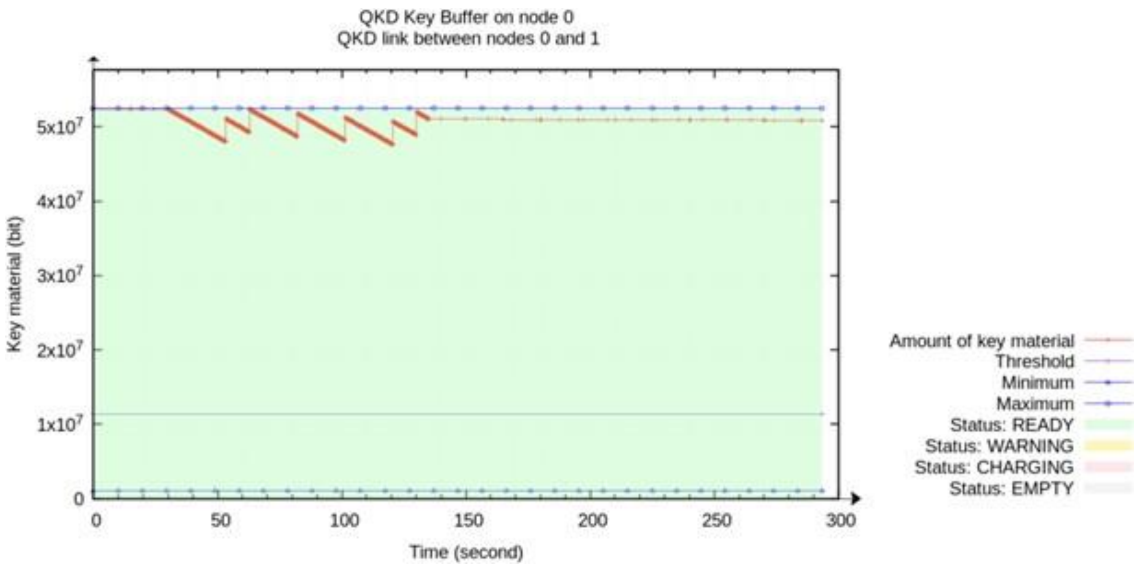


Figure 4.19: QKD buffer capacity between nodes 0 and 1

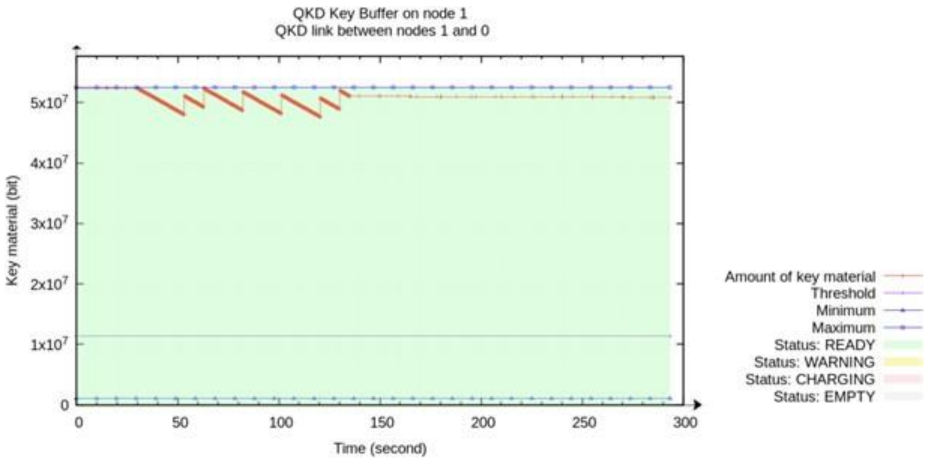


Figure 4.20: QKD buffer capacity between nodes 1 and 0

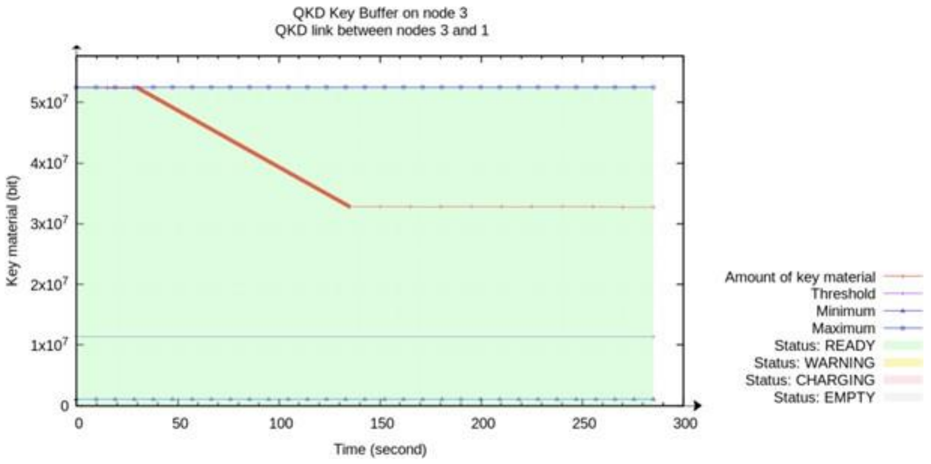


Figure 4.21: QKD buffer capacity between nodes 3 and 1

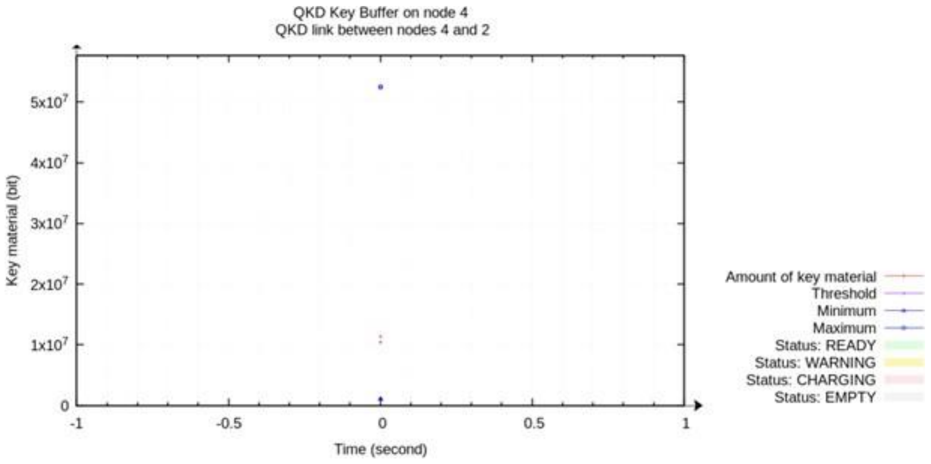
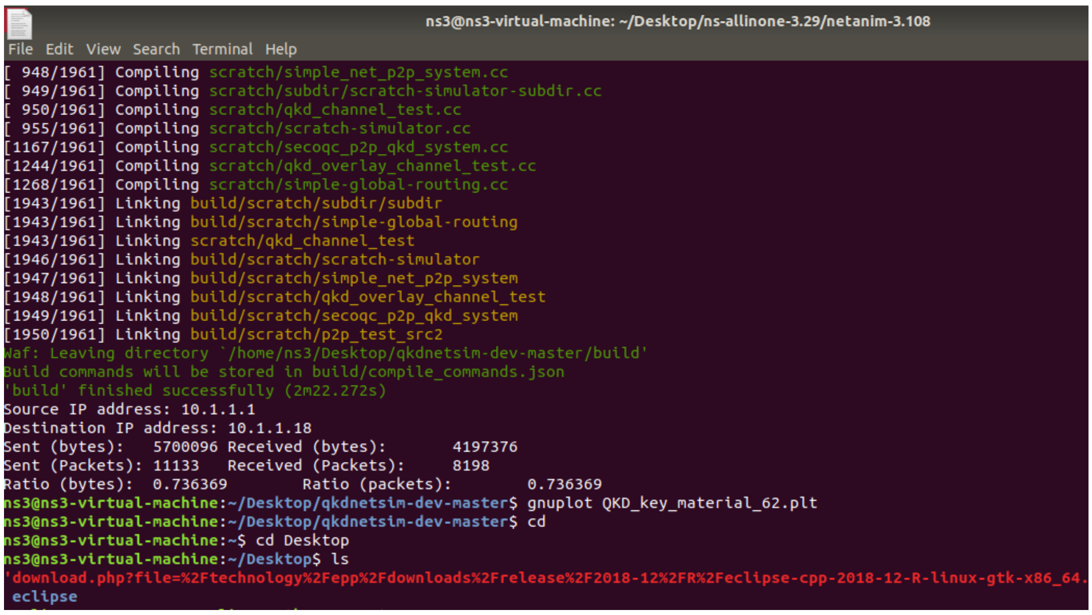


Figure 4.22: QKD buffer capacity between nodes 4 and 2

4. Eight node overlay channel implementation for peer to peer network using ipv4 (eighttestqkd.cc) This example displays the implementation of a complex network that relays information in the quantum channel. The QKD buffer graphs are not ideal due to channel disruption. It also includes a routing table based on Destination-Sequenced Distance-Vector Routing (DSDV), which explains data transfer via IPv4.



ns3@ns3-virtual-machine: ~/Desktop/ns-allinone-3.29/netanim-3.108
File Edit View Search Terminal Help

```
[ 948/1961] Compiling scratch/simple_net_p2p_system.cc
[ 949/1961] Compiling scratch/subdir/scratch-simulator-subdir.cc
[ 950/1961] Compiling scratch/qkd_channel_test.cc
[ 955/1961] Compiling scratch/scratch-simulator.cc
[1167/1961] Compiling scratch/secoqc_p2p_qkd_system.cc
[1244/1961] Compiling scratch/qkd_overlay_channel_test.cc
[1268/1961] Compiling scratch/simple-global-routing.cc
[1943/1961] Linking build/scratch/subdir/subdir
[1943/1961] Linking build/scratch/simple-global-routing
[1943/1961] Linking scratch/qkd_channel_test
[1946/1961] Linking build/scratch/scratch-simulator
[1947/1961] Linking build/scratch/simple_net_p2p_system
[1948/1961] Linking build/scratch/qkd_overlay_channel_test
[1949/1961] Linking build/scratch/secoqc_p2p_qkd_system
[1950/1961] Linking build/scratch/p2p_test_src2
Waf: Leaving directory `/home/ns3/Desktop/qkdnetsim-dev-master/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (2m22.272s)
Source IP address: 10.1.1.1
Destination IP address: 10.1.1.18
Sent (bytes):    5700096 Received (bytes):       4197376
Sent (Packets): 11133   Received (Packets):     8198
Ratio (bytes):  0.736369        Ratio (packets):        0.736369
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master$ gnuplot QKD_key_material_62.plt
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master$ cd
ns3@ns3-virtual-machine:~$ cd Desktop
ns3@ns3-virtual-machine:~/Desktop$ ls
'download.php?file=%2Ftechnology%2Fepp%2Fdownloads%2Frelease%2F2018-12%2FR%2Feclipse-cpp-2018-12-R-linux-gtk-x86_64.
 eclipse
```

Figure 4.23: Output for eighttestqkd.cc

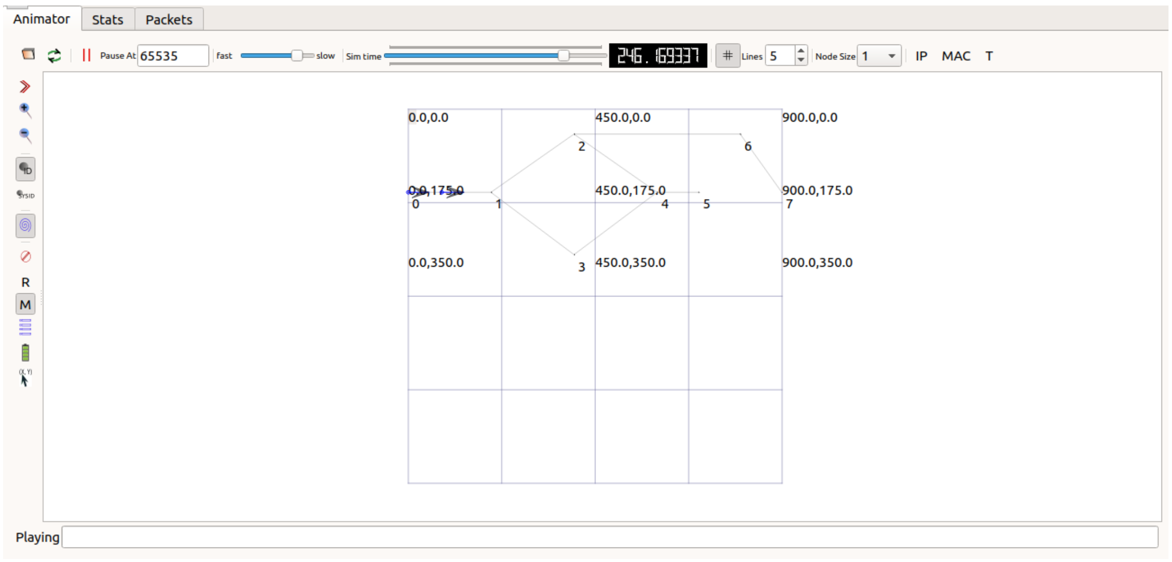


Figure 4.24: eighttestqkd.cc network animation

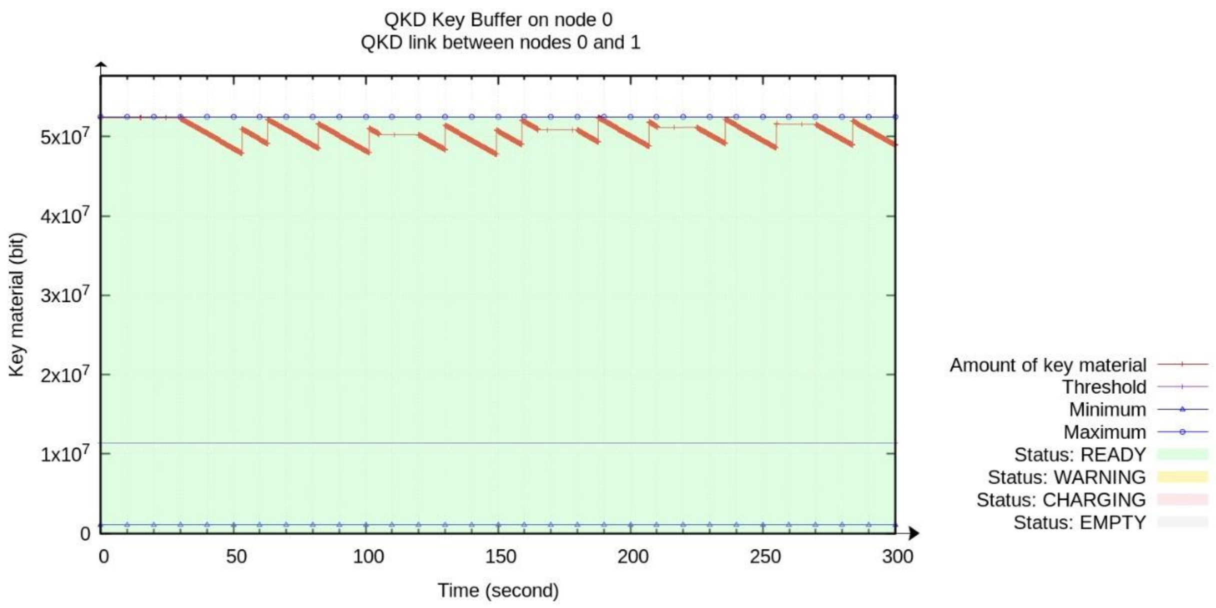


Figure 4.25: QKD buffer capacity between nodes 0 and 1

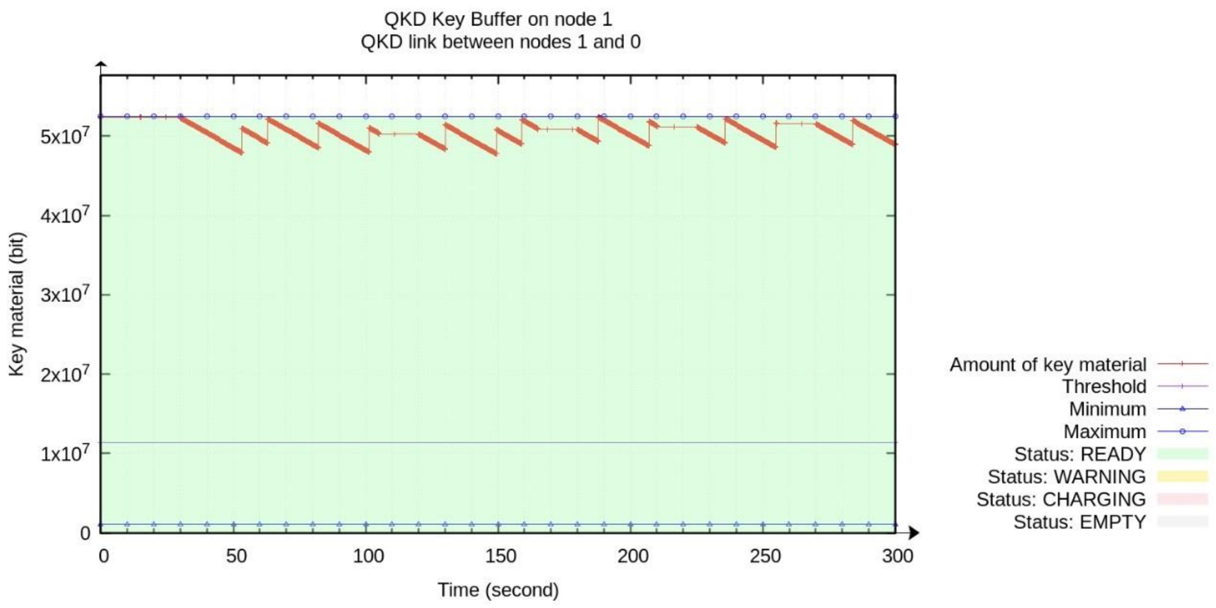


Figure 4.26: QKD buffer capacity between nodes 1 and 0

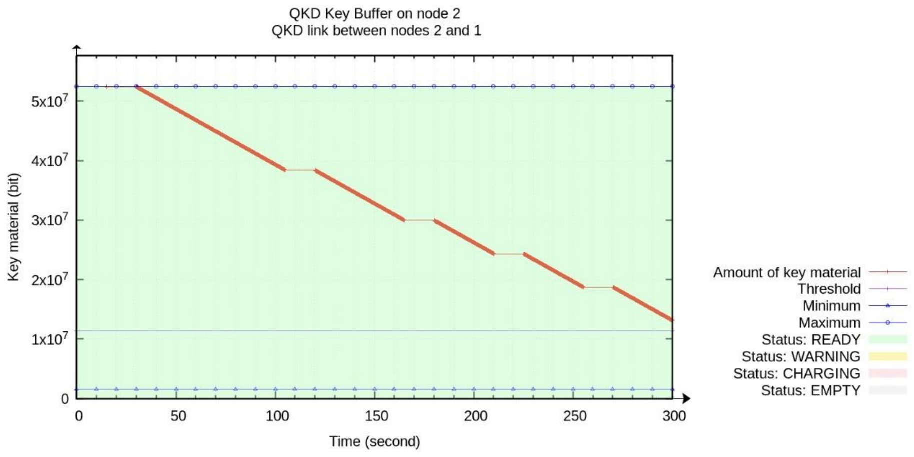


Figure 4.27: QKD buffer capacity between nodes 2 and 1

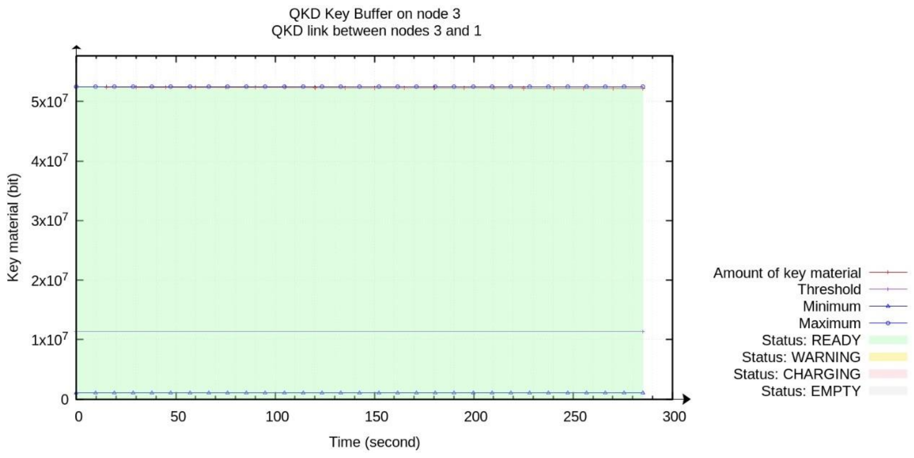


Figure 4.28: QKD buffer capacity between nodes 3 and 1

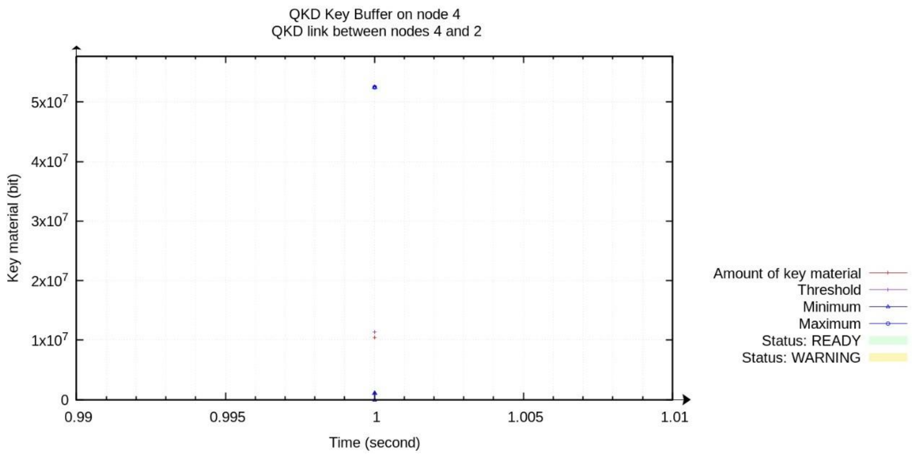


Figure 4.29: QKD buffer capacity between nodes 4 and 2

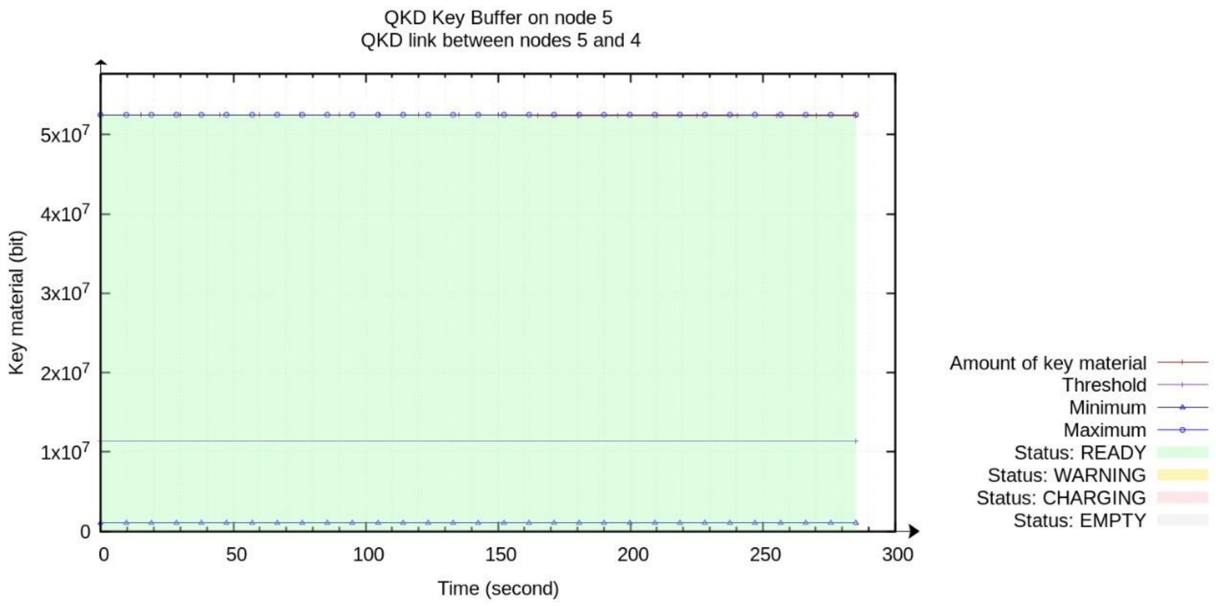


Figure 4.30: QKD buffer capacity between nodes 5 and 4

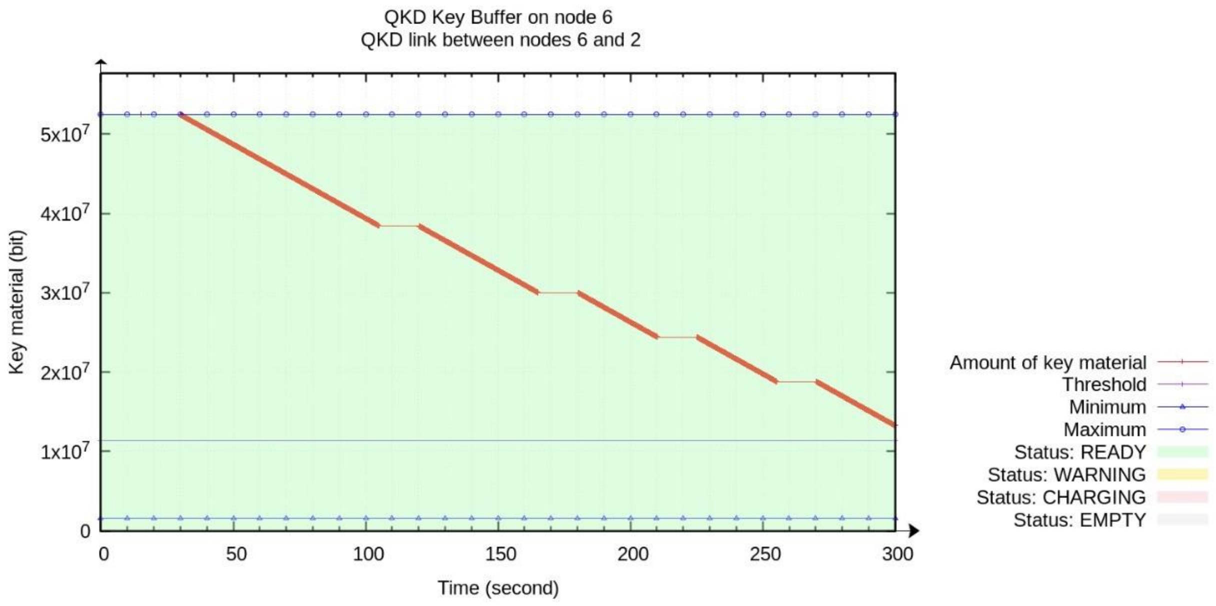
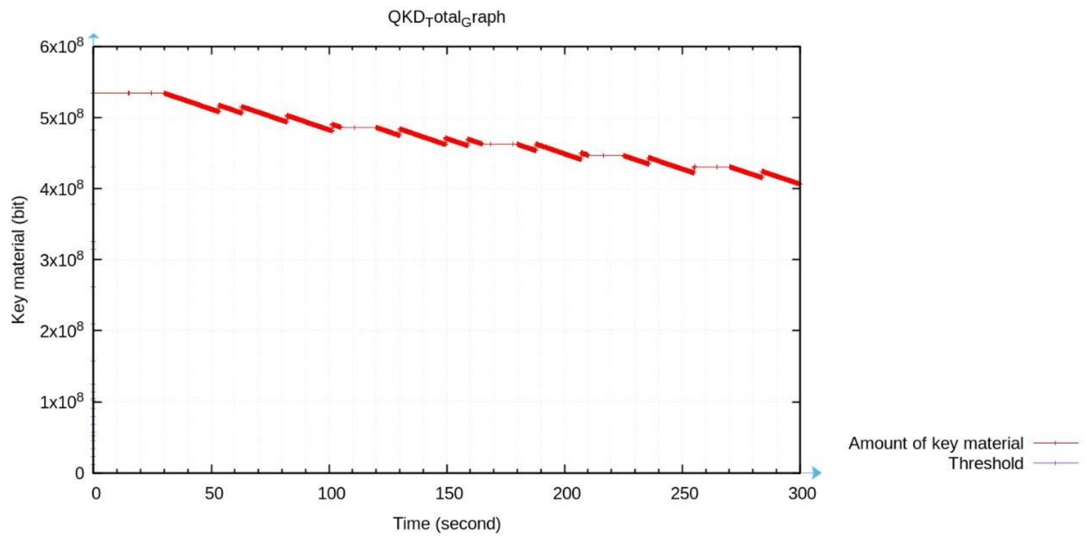
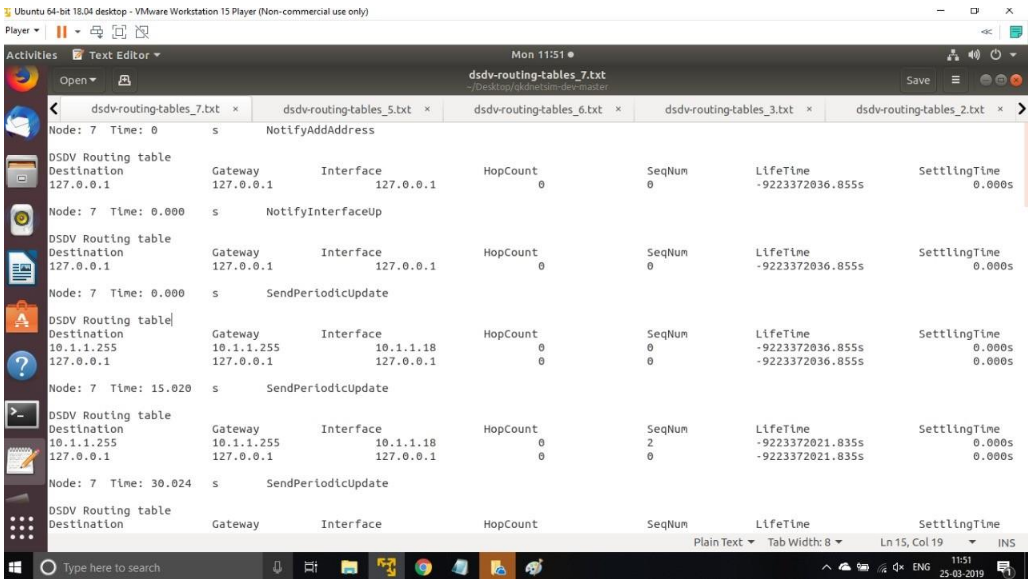


Figure 4.31: QKD buffer capacity between nodes 6 and 2



graph.jpeg

Figure 4.32: Total graph for QKD buffer capacity

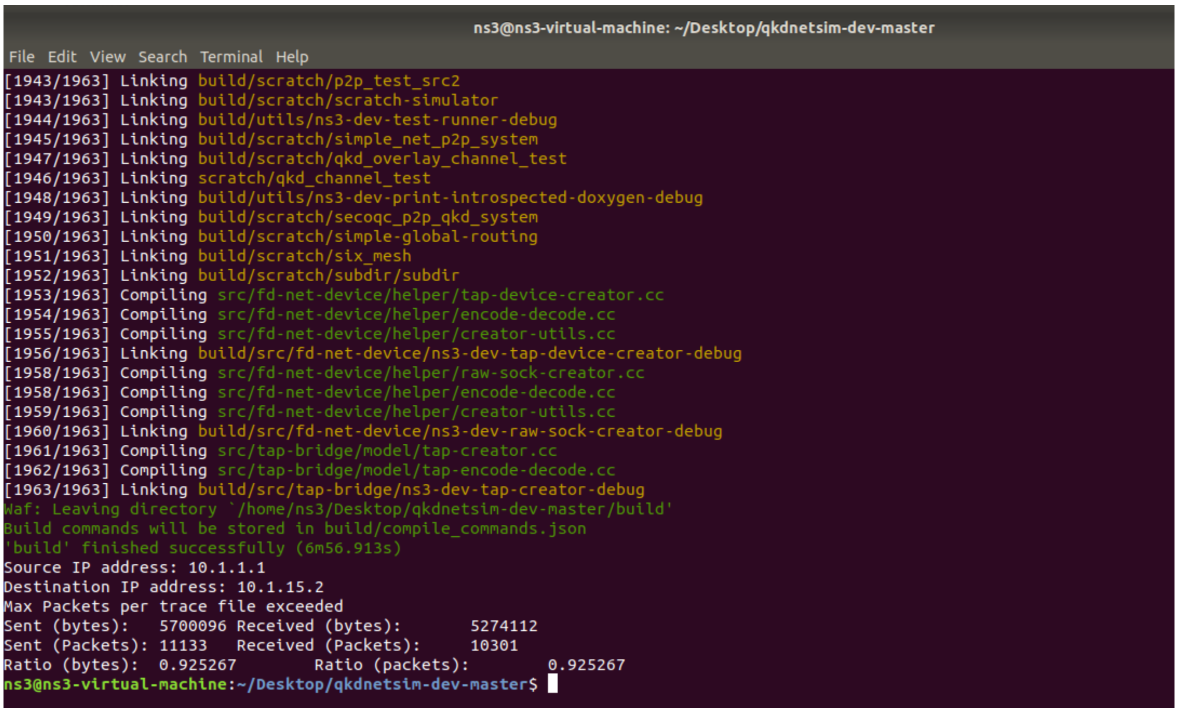

Ubuntu 64-bit 18.04 desktop - VMware Workstation 15 Player (Non-commercial use only)
Player
Activities
Text Editor
Mon 11:51
Open
dsdv-routing-tables_7.txt
~/Desktop/qkdnetsim-dev-master
Save
dsdv-routing-tables_7.txt
dsdv-routing-tables_5.txt
dsdv-routing-tables_6.txt
dsdv-routing-tables_3.txt
dsdv-routing-tables_2.txt
Plain Text
Tab Width: 8
Ln 15, Col 19
INS
Type here to search
ENG
11:51
25-03-2019

```
Node: 7  Time: 0          s       NotifyAddAddress

DSDV Routing table
Destination             Gateway         Interface               HopCount                SeqNum          LifeTime                SettlingTime
127.0.0.1               127.0.0.1               127.0.0.1               0               0               -9223372036.855s                0.000s

Node: 7  Time: 0.000      s       NotifyInterfaceUp

DSDV Routing table
Destination             Gateway         Interface               HopCount                SeqNum          LifeTime                SettlingTime
127.0.0.1               127.0.0.1               127.0.0.1               0               0               -9223372036.855s                0.000s

Node: 7  Time: 0.000      s       SendPeriodicUpdate

DSDV Routing table
Destination             Gateway         Interface               HopCount                SeqNum          LifeTime                SettlingTime
10.1.1.255              10.1.1.255              10.1.1.18               0               0               -9223372036.855s                0.000s
127.0.0.1               127.0.0.1               127.0.0.1               0               0               -9223372036.855s                0.000s

Node: 7  Time: 15.020     s       SendPeriodicUpdate

DSDV Routing table
Destination             Gateway         Interface               HopCount                SeqNum          LifeTime                SettlingTime
10.1.1.255              10.1.1.255              10.1.1.18               0               2               -9223372021.835s                0.000s
127.0.0.1               127.0.0.1               127.0.0.1               0               0               -9223372021.835s                0.000s

Node: 7  Time: 30.024     s       SendPeriodicUpdate

DSDV Routing table
Destination             Gateway         Interface               HopCount                SeqNum          LifeTime                SettlingTime
```

Figure 4.33: Routing table for node 7

5. Five node mesh network overlay channel implementations for peer to peer network using ipv4 (sixmesh.cc)

This example displays the implementation of a mesh network that relays information in the quantum channel. The QKD buffer graphs are not ideal due to channel disruption. It also includes a routing table based on Destination-Sequenced Distance-Vector Routing (DSDV), which explains data transfer via IPv4.


ns3@ns3-virtual-machine: ~/Desktop/qkdnetsim-dev-master
File Edit View Search Terminal Help

```
[1943/1963] Linking build/scratch/p2p_test_src2
[1943/1963] Linking build/scratch/scratch-simulator
[1944/1963] Linking build/utils/ns3-dev-test-runner-debug
[1945/1963] Linking build/scratch/simple_net_p2p_system
[1947/1963] Linking build/scratch/qkd_overlay_channel_test
[1946/1963] Linking scratch/qkd_channel_test
[1948/1963] Linking build/utils/ns3-dev-print-introspected-doxygen-debug
[1949/1963] Linking build/scratch/secoqc_p2p_qkd_system
[1950/1963] Linking build/scratch/simple-global-routing
[1951/1963] Linking build/scratch/six_mesh
[1952/1963] Linking build/scratch/subdir/subdir
[1953/1963] Compiling src/fd-net-device/helper/tap-device-creator.cc
[1954/1963] Compiling src/fd-net-device/helper/encode-decode.cc
[1955/1963] Compiling src/fd-net-device/helper/creator-utils.cc
[1956/1963] Linking build/src/fd-net-device/ns3-dev-tap-device-creator-debug
[1958/1963] Compiling src/fd-net-device/helper/raw-sock-creator.cc
[1958/1963] Compiling src/fd-net-device/helper/encode-decode.cc
[1959/1963] Compiling src/fd-net-device/helper/creator-utils.cc
[1960/1963] Linking build/src/fd-net-device/ns3-dev-raw-sock-creator-debug
[1961/1963] Compiling src/tap-bridge/model/tap-creator.cc
[1962/1963] Compiling src/tap-bridge/model/tap-encode-decode.cc
[1963/1963] Linking build/src/tap-bridge/ns3-dev-tap-creator-debug
Waf: Leaving directory `/home/ns3/Desktop/qkdnetsim-dev-master/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (6m56.913s)
Source IP address: 10.1.1.1
Destination IP address: 10.1.15.2
Max Packets per trace file exceeded
Sent (bytes):   5700096 Received (bytes):       5274112
Sent (Packets): 11133   Received (Packets):     10301
Ratio (bytes):  0.925267        Ratio (packets):        0.925267
ns3@ns3-virtual-machine:~/Desktop/qkdnetsim-dev-master$
```

Figure 4.34: Output for sixmesh.cc

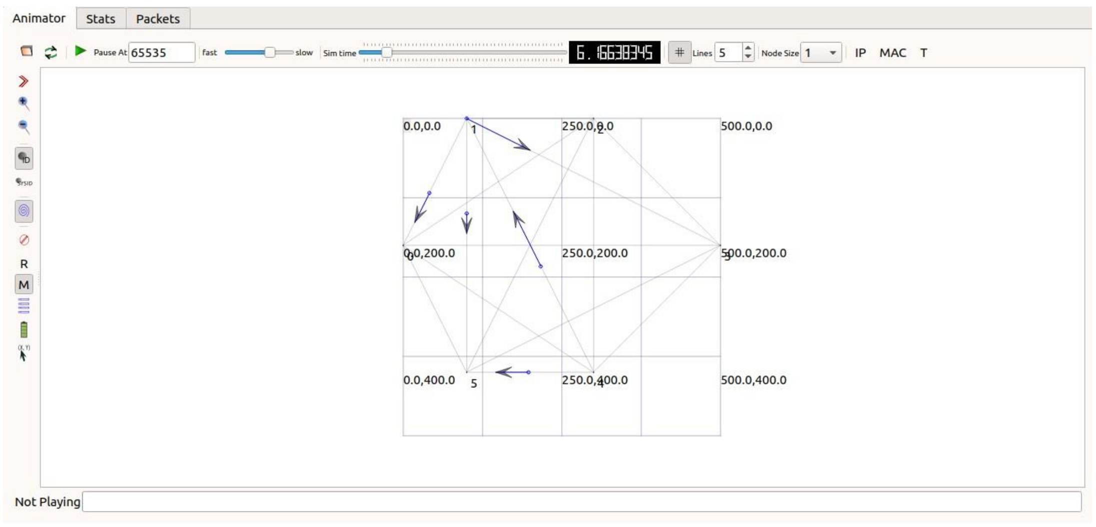


Figure 4.35: Network topology for sixmesh.cc

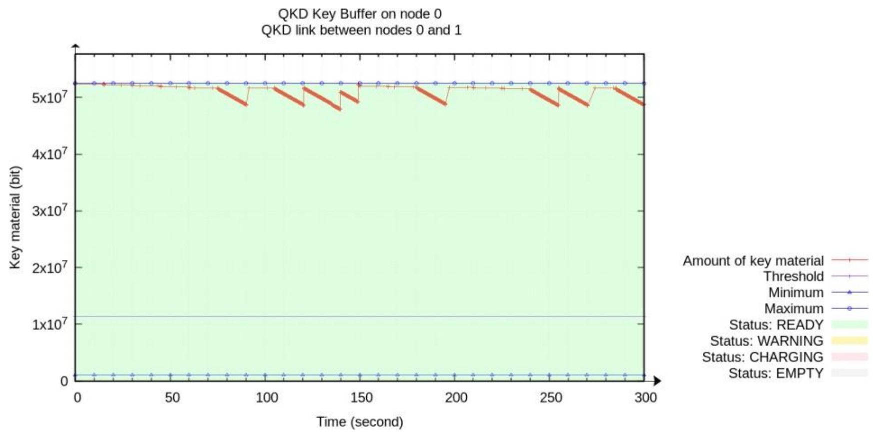


Figure 4.36: QKD buffer capacity between nodes 0 and 1

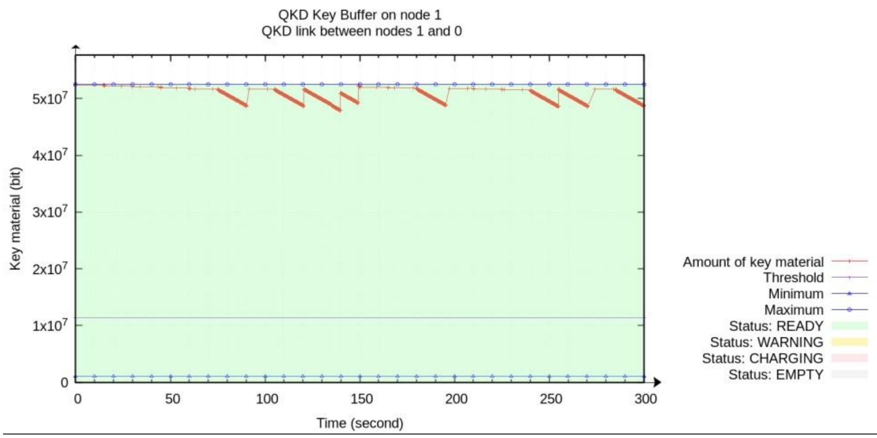


Figure 4.37: QKD buffer capacity between nodes 1 and 0

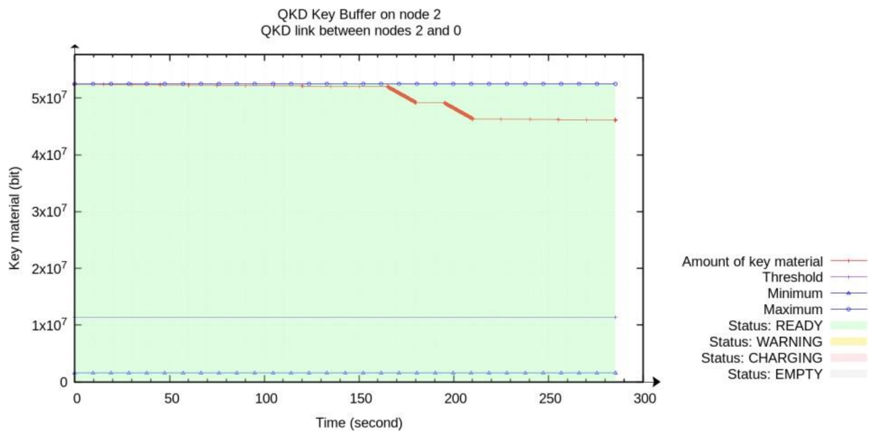


Figure 4.38: QKD buffer capacity between nodes 2 and 0

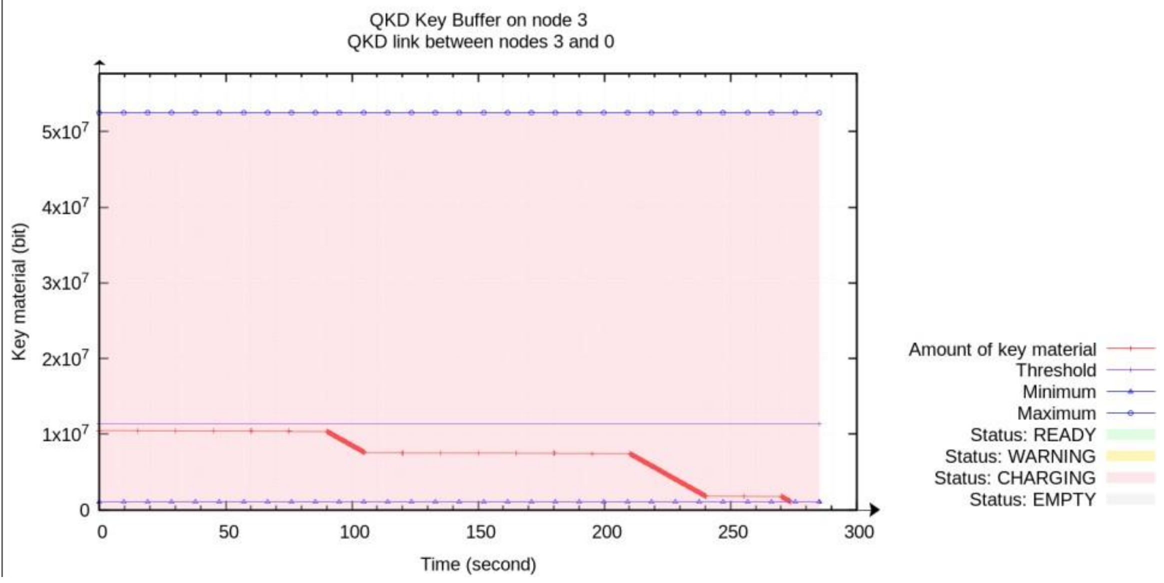


Figure 4.39: QKD buffer capacity between nodes 3 and 0

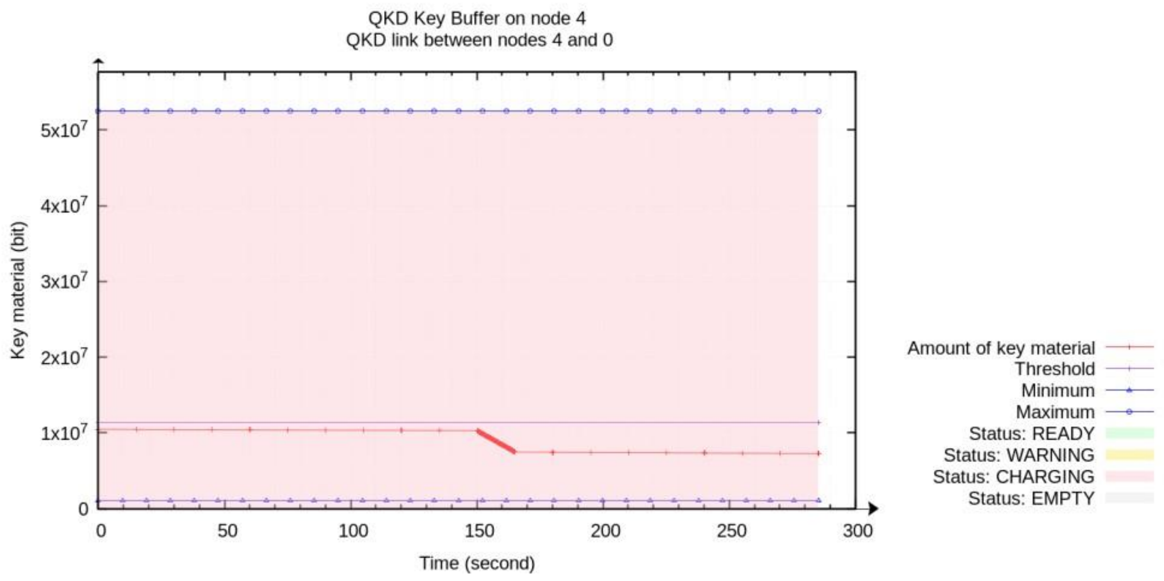


Figure 4.40: QKD buffer capacity between nodes 4 and 0

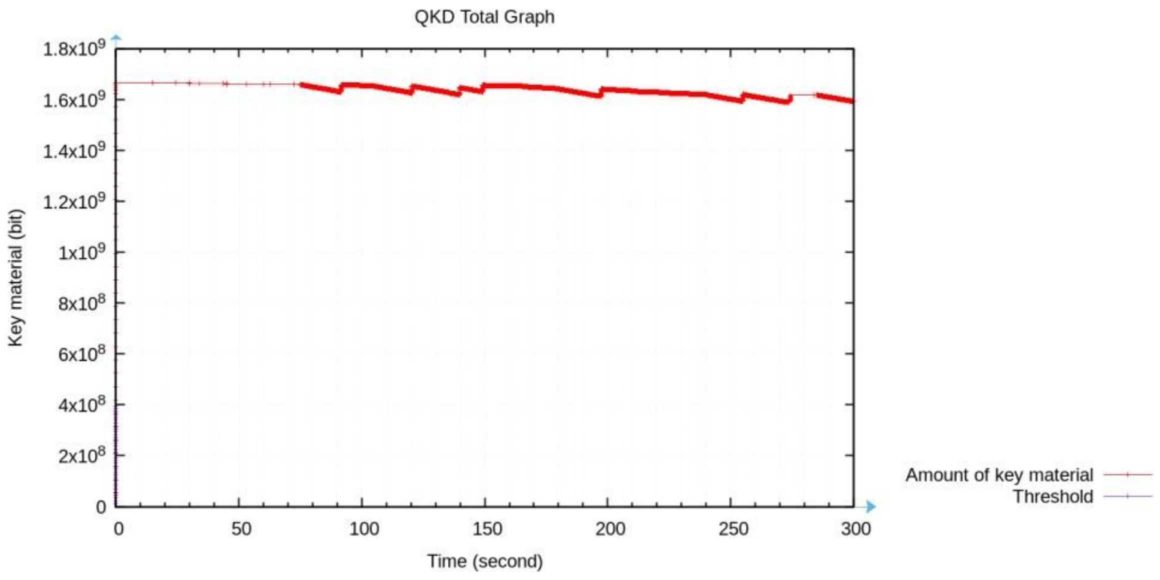
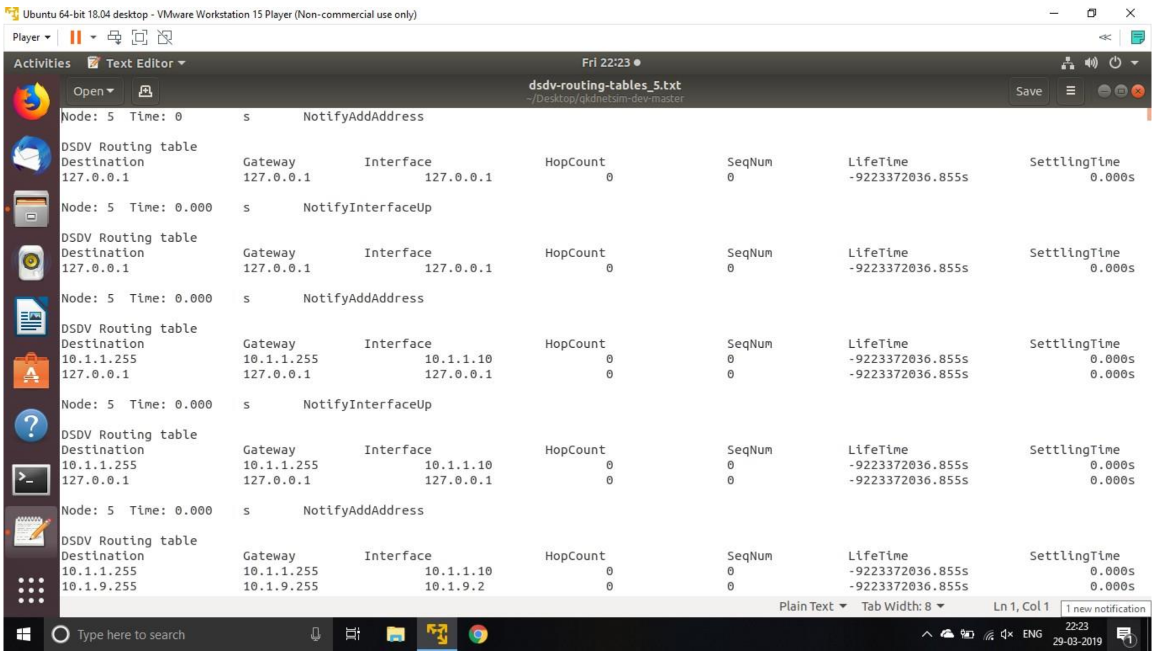


Figure 4.41: Total QKD buffer capacity graph


Ubuntu 64-bit 18.04 desktop - VMware Workstation 15 Player (Non-commercial use only)
Player
Activities
Text Editor
Fri 22:23
Open
dsdv-routing-tables_5.txt
~/Desktop/qkdnetsim-dev-master
Save
Plain Text
Tab Width: 8
Ln 1, Col 1
1 new notification
Type here to search
ENG
22:23
29-03-2019

```
Node: 5  Time: 0        s       NotifyAddAddress

DSDV Routing table
Destination             Gateway         Interface               HopCount                SeqNum          LifeTime                SettlingTime
127.0.0.1               127.0.0.1               127.0.0.1               0               0               -9223372036.855s                0.000s

Node: 5  Time: 0.000    s       NotifyInterfaceUp

DSDV Routing table
Destination             Gateway         Interface               HopCount                SeqNum          LifeTime                SettlingTime
127.0.0.1               127.0.0.1               127.0.0.1               0               0               -9223372036.855s                0.000s

Node: 5  Time: 0.000    s       NotifyAddAddress

DSDV Routing table
Destination             Gateway         Interface               HopCount                SeqNum          LifeTime                SettlingTime
10.1.1.255              10.1.1.255              10.1.1.10               0               0               -9223372036.855s                0.000s
127.0.0.1               127.0.0.1               127.0.0.1               0               0               -9223372036.855s                0.000s

Node: 5  Time: 0.000    s       NotifyInterfaceUp

DSDV Routing table
Destination             Gateway         Interface               HopCount                SeqNum          LifeTime                SettlingTime
10.1.1.255              10.1.1.255              10.1.1.10               0               0               -9223372036.855s                0.000s
127.0.0.1               127.0.0.1               127.0.0.1               0               0               -9223372036.855s                0.000s

Node: 5  Time: 0.000    s       NotifyAddAddress

DSDV Routing table
Destination             Gateway         Interface               HopCount                SeqNum          LifeTime                SettlingTime
10.1.1.255              10.1.1.255              10.1.1.10               0               0               -9223372036.855s                0.000s
10.1.9.255              10.1.9.255              10.1.9.2                0               0               -9223372036.855s                0.000s
```

Figure 4.42: Routing table for node 5

CHAPTER 5

CONCLUSION

This project deals with the practical realization of QKD networks from a network point of view. The main part of this project deals with the QKDNetSim simulation environment which is primarily intended for testing of the existing solutions and the implementation of the new solutions in the field of QKD networks. It looks into the different quantum key distribution protocols and as well as the various available soft-ware's available for QKD simulation. The work summarizes the limitations and the basic characteristics of a QKD network and describes the ways of implementing QKD networks, which can be either in an overlay mode or as a network with a single TCP/IP stack. We have also analyzed the factors that affect traffic in the quantum channel.

The implementation of this project has been done in two parts, firstly we havetried to implement a normal public channel at ideal conditions as well as a public channel network with congestion control. Secondly, we have tried to run a simple quantum channel network at ideal conditions, after which we have tried to implement a more complicated overlay network of the same to identify the changes in the channel traffic. Taking into account the various modules present in QKDNetSim, we observed that the QKD buffer capacity graph enables us to analyze the traffic in the current path of the network, and choose a more efficient path for data exchange. A dip in the QKD buffer capacity indicates that parts of the key have been dropped. This can be due to congestion, which is unlikely in a point to point network hence allowing us to conclude that it indicates the presence on an intruder in the path.

This simulation has been done by utilizing the BB84 protocol due to its ease of implementation. Although other protocols can also be implemented within the developed simulation environment. The simulations carried out in this paper have been done using NS-3 with QKDNetSim built into it. Hence, the main purpose of this project was to prove theoretical concepts in the form of a simulation which closely emulates the real- life constrains of data transmission. Thus, proving that QKD is the most secure means of data encryption.

Multiple advancements that have been made in the field of quantum cryptography even though they are purely hypothesis based, they depict the path in which this research is headed. The transmission of data in the quantum channel is done by means of fiber optics, hence the it is affected by the problems that exist in this form of data transmission. There is research that exists that counters such problems by referencing frame independent QKD protocol for polarization qubits and hence showing that it overcomes detrimental effects of drifting fiber birefringence in a polarization maintaining fiber. Some research also suggests that all-graphene solid-state components can be used within quantum networks, due to their possible implementation as quantum repeaters. There is also the possibility of implementing satellite to ground quantum key distribution based on the properties of entanglement. A theoretical approach in the implementation of QKD in IoT to increase the security of wireless sensor-based networks as well as cellular networks has been researched. It is also possible to further enhance the existing software resources to help implement more uncertainty principle-based protocols as well as test entanglement-based protocols. Also, the level of complexity of the networks can be increased, multiple constraints can be added to the simulated networks, so that it can more closely emulate a real-life scenario. Since the current research done in this field is purely theoretical, the simulation constraints can be further enhanced to implement those research guidelines. Some guidelines could be QKD using quantum gates, QKD over multiple terrains, QKD transmission over wireless networks, encrypting cellular transmission using QKD. Hence, the developed model can be further enhanced by adding the following features:

1. Implement an entanglement-based system that can be utilized for long distance com- munication
2. Adding congestion control features to the quantum channel in order to implement more complicated networks in the quantum channel
3. Introducing more variables in the simulation to create a model that more closely emulates real-life condition.

REFERENCES

[1] Mehic, Miralem, Oliver Maurhart, Stefan Rass, and Miroslav Voznak. "**Implementation of quantum key distribution network simulation module in the network simulator NS-3**." Quantum Information Processing 16, no. 10 (2017): 253.

[2] Morris, Jeffrey D., Douglas D. Hodson, Michael R. Grimaila, David R. Jacques, and Gerald Baumgartner. "**Towards the modeling and simulation of quantum key distribu- tion systems**." International Journal of Emerging Technology and Advanced Engineer- ing 4, no. 2 (2014): 829-838

[3] Abushgra, Abdulbast, and Khaled Elleithy. "**QKDP's comparison based upon quan- tum cryptography rules**." In 2016 IEEE Long Island Systems, Applications and Tech- nology Conference (LISAT), pp. 1-5. IEEE, 2016.

[4] Jasim, Omer K., Safia Abbas, El-Sayed M. El-Horbaty, and Abdel-Badeeh M. Salem. "**Quantum key distribution: simulation and characterizations**." Procedia Computer Science 65, 701-710, 2015.

[5] Padamvathi, V., B. Vishnu Vardhan, and A. V. N. Krishna. "**Quantum Cryptography and Quantum Key Distribution Protocols: A Survey**." In 2016 IEEE 6th International Conference on Advanced Computing (IACC), pp. 556-562. IEEE, 2016.

[6] Trizna, Anastasija, and Andris Ozols. "**An Overview of Quantum Key Distribution Protocols**." Information Technology Management Science 21, 2018.

[7] Alshowkan, Muneer, and Khaled M. Elleithy. "**Deterministic and Efficient Quantum Key Distribution Using Entanglement Parity Bits and Ancillary Qubits**." IEEE Access 5, 2017: 25565-25575.

[8] Goldman, Jeremy. "**Quantum Cryptography–Current Methods and Technology**." (2014).

[9] Fröhlich, Bernd, Marco Lucamarini, James F. Dynes, Lucian C. Comandar, Winci W-S. Tam, Alan Plews, Andrew W. Sharpe, Zhiliang Yuan, and Andrew J. Shields. "**Long-distance quantum key distribution secure against coherent attacks**." Optica 4, no. 1, 2017: 163-167.

[10] Lobino, Mirko, Pei Zhang, Enrique Martín-López, Richard W. Nock, Damien Bonneau, Hong Wei Li, Antti O. Niskanen et al. "**Quantum key distribution with integrated optics**." In 2014 19th Asia and South Pacific Design Automation Conference (ASP- DAC), pp. 795-799. IEEE, 2014.

[11] Zhong, Bo, Weiyue Liu, Chong Li, Zhidi Jiang, Yanlin Tang, Yang Liu, Bin Yang, Xiaofang Hu, Qi Shen, and Chenzhi Peng. "**Qubit Extraction for Free-Space Quantum Key Distribution**." In 2011 International Conference on Network Computing and Infor- mation Security, vol. 1, pp. 242-244. IEEE, 2011.

[12] Wu, G. Y., and N-Y. Lue. "**Graphene-based qubits in quantum communications**." Physical Review B 86, no. 4 (2012): 045456.

[13] Yin, Juan, Yuan Cao, Yu-Huai Li, Ji-Gang Ren, Sheng-Kai Liao, Liang Zhang, Wen-Qi Cai et al. "**Satellite-to-ground entanglement-based quantum key distribution**."

Physical review letters 119, no. 20 (2017): 200501

[14] Mehic, Miralem, Peppino Fazio, Miroslav Vozňák, and Erik Chromý. "**Toward designing a quantum key distribution network simulation model**." Information and Com- munication Technologies and Services (2016).

[15] Routray, Sudhir K., Mahesh K. Jha, Laxmi Sharma, Rahul Nyamangoudar, Abhishek Javali, and Sutapa Sarkar. "**Quantum cryptography for IoT: A Perspective**." In 2017 International Conference on IoT and Application (ICIOT), pp. 1-4. IEEE, 2017.

[16] Niemiec, Marcin, Łukasz Romański, and Marcin Święty. "**Quantum cryptography protocol simulator."** In International Conference on Multimedia Communications, Services and Security, pp. 286-292. Springer, Berlin, Heidelberg, 2011.

APPENDIX

Program

1. Public channel implementation (first.cc)

```
 #include "ns3/core-module.h"
 #include "ns3/network-module.h"
 #include "ns3/internet-module.h"
 #include "ns3/point-to-point-module.h"
 #include "ns3/applications-module.h"
 #include "ns3/netanim-module.h"

 using namespace ns3;
 NS_LOG_COMPONENT_DEFINE
("FirstScriptExample");

 int
 main (int argc, char *argv[])
 {
  CommandLine cmd;
  cmd.Parse (argc, argv);

  Time::SetResolution (Time::NS);
  LogComponentEnable
("UdpEchoClientApplication",
LOG_LEVEL_INFO);
  LogComponentEnable
("UdpEchoServerApplication",
LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute
("DataRate", StringValue ("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay",
StringValue ("2ms"));

  NetDeviceContainer devices;
  devices = pointToPoint.Install (nodes);
  InternetStackHelper stack;
  stack.Install (nodes);

Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0",
"255.255.255.0");

  Ipv4InterfaceContainer interfaces =
address.Assign (devices);

  UdpEchoServerHelper echoServer (9);
```

```
 ApplicationContainer serverApps =
echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient
(interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets",
UintegerValue (1));
  echoClient.SetAttribute ("Interval",
TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize",
UintegerValue (1024));

  ApplicationContainer clientApps =
echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  AnimationInterface anim ("anim1.xml");
  anim.SetConstantPosition(nodes.Get(0), 1.0,
2.0);
  anim.SetConstantPosition(nodes.Get(1), 10.0,
10.0);

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

2. Quantum channel implementation (qkdchanneltest.cc)

```
// Network topology
//
//      n0 ---p2p-- n1 --p2p-- n2
//       |---------qkd---------|
//
// - udp flows from n0 to n2

#include <fstream>
#include "ns3/core-module.h"
#include "ns3/applications-module.h"
#include "ns3/internet-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/mobility-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/gnuplot.h"

#include "ns3/qkd-helper.h"
#include "ns3/qkd-app-charging-
helper.h"
#include "ns3/qkd-send.h"
#include "ns3/aodv-module.h"
#include "ns3/olsr-module.h"
#include "ns3/dsdv-module.h"

#include "ns3/network-module.h"
#include "ns3/fd-net-device-module.h"
#include "ns3/internet-apps-module.h"
#include "ns3/netanim-module.h"

#include "crypto++/aes.h"
#include "crypto++/modes.h"
#include "crypto++/filters.h"
#include <sstream>
#include <string>

using namespace ns3;
NS_LOG_COMPONENT_DEFINE
("QKD_CHANNEL_TEST");
uint32_t m_bytes_total = 0;

uint32_t m_bytes_received = 0;
uint32_t m_bytes_sent = 0;
uint32_t m_packets_received = 0;
double m_time = 0;
void
SentPacket(std::string context,
Ptr<const Packet> p){
   m_bytes_sent += p->GetSize();
```

```
void
ReceivedPacket(std::string context,
Ptr<const Packet> p, const Address&
addr){

m_bytes_received += p->GetSize();
   m_bytes_total += p->GetSize();
   m_packets_received++;

}
Ratio(uint32_t m_bytes_sent, uint32_t
m_packets_sent ){
   std::cout << "Sent (bytes):\t" <<
m_bytes_sent
   << "\tReceived (bytes):\t" <<
m_bytes_received
   << "\nSent (Packets):\t" <<
m_packets_sent
   << "\tReceived (Packets):\t" <<
m_packets_received
<< "\nRatio (bytes):\t" <<
(float)m_bytes_received/(float)m_bytes_s
ent
   << "\tRatio (packets):\t" <<
(float)m_packets_received/(float)m_packe
ts_sent << "\n";
}
void
int main (int argc, char *argv[])
{
   Packet::EnablePrinting();
   PacketMetadata::Enable ();
   //
   // Explicitly create the nodes required
by the topology (shown above).
   //
   NS_LOG_INFO ("Create nodes.");
   NodeContainer n;
   n.Create (3);

   NodeContainer n0n1 = NodeContainer
(n.Get(0), n.Get (1));
   NodeContainer n1n2 = NodeContainer
(n.Get(1), n.Get (2));
//Enable OLSR
   //AodvHelper routingProtocol;
   //OlsrHelper routingProtocol;
   DsdvHelper routingProtocol;
   InternetStackHelper internet;
   internet.SetRoutingHelper
(routingProtocol);
   internet.Install (n);

   // Set Mobility for all nodes
   MobilityHelper mobility;
   Ptr<ListPositionAllocator>
positionAlloc = CreateObject
<ListPositionAllocator>();
   positionAlloc ->Add(Vector(0, 200, 0));
// node0
   positionAlloc ->Add(Vector(200, 200,
0)); // node1
   positionAlloc ->Add(Vector(400, 200,
0)); // node2
mobility.SetPositionAllocator(positionAll
oc);

mobility.SetMobilityModel("ns3::Constan
tPositionMobilityModel");
   mobility.Install(n);

   // We create the channels first without
any IP addressing information
NS_LOG_INFO ("Create channels.");
   PointToPointHelper p2p;
   p2p.SetDeviceAttribute ("DataRate",
StringValue ("5Mbps"));
   p2p.SetChannelAttribute ("Delay",
StringValue ("2ms"));

   NetDeviceContainer d0d1 = p2p.Install
(n0n1);
   NetDeviceContainer d1d2 = p2p.Install
(n1n2);

   //
   // We've got the "hardware" in place.
Now we need to add IP addresses.
   //
   NS_LOG_INFO ("Assign IP
Addresses.");
   Ipv4AddressHelper ipv4;

   ipv4.SetBase ("10.1.1.0",
"255.255.255.0");
   Ipv4InterfaceContainer i0i1 =
ipv4.Assign (d0d1);
```

```
   ipv4.SetBase ("10.1.2.0",
"255.255.255.0");
   Ipv4InterfaceContainer i1i2 =
ipv4.Assign (d1d2);

   //
   // Explicitly create the channels
required by the topology (shown above).
   //
   //  install QKD Managers on the nodes
   QKDHelper QHelper;
   QHelper.InstallQKDManager (n);


   //create QKD connection between nodes
0 and 1
   NetDeviceContainer qkdNetDevices01
= QHelper.InstallQKD (
      d0d1.Get(0), d0d1.Get(1),
      1048576,    //min
      11324620, //thr
      52428800,   //max
      52428800     //current    //20485770
   );

   //Create graph to monitor buffer
changes
   QHelper.AddGraph(n.Get(0),
d0d1.Get(0)); //srcNode,
destinationAddress, BufferTitle


   //create QKD connection between nodes
1 and 2
   NetDeviceContainer qkdNetDevices12
= QHelper.InstallQKD (
      d1d2.Get(0), d1d2.Get(1),
      1048576,    //min
      11324620, //thr
      52428800,   //max
      52428800     //current    //20485770
   );
   //Create graph to monitor buffer
changes
   QHelper.AddGraph(n.Get(1),
d0d1.Get(0)); //srcNode,
destinationAddress, BufferTitle
   NS_LOG_INFO ("Create
Applications.");
```

```
    std::cout << "Source IP address: " <<
i0i1.GetAddress(0) << std::endl;
   std::cout << "Destination IP address: "
<< i1i2.GetAddress(1) << std::endl;

   /* QKD APPs for charing  */
   QKDAppChargingHelper
qkdChargingApp("ns3::TcpSocketFactory
", i0i1.GetAddress(0),
i0i1.GetAddress(1), 3072000);
   ApplicationContainer qkdChrgApps =
qkdChargingApp.Install ( d0d1.Get(0),
d0d1.Get(1) );
   qkdChrgApps.Start (Seconds (5.));
   qkdChrgApps.Stop (Seconds (1500.));
QKDAppChargingHelper
qkdChargingApp12("ns3::TcpSocketFacto
ry", i1i2.GetAddress(0),
i1i2.GetAddress(1), 3072000);
   ApplicationContainer qkdChrgApps12
= qkdChargingApp12.Install (
d1d2.Get(0), d1d2.Get(1) );
   qkdChrgApps12.Start (Seconds (5.));
   qkdChrgApps12.Stop (Seconds
(1500.));


   /* Create user's traffic between v0 and
v1 */
   /* Create sink app */
   uint16_t sinkPort = 8080;
   QKDSinkAppHelper packetSinkHelper
("ns3::UdpSocketFactory",
InetSocketAddress (Ipv4Address::GetAny
(), sinkPort));
   ApplicationContainer sinkApps =
packetSinkHelper.Install (n.Get (2));
   sinkApps.Start (Seconds (25.));
   sinkApps.Stop (Seconds (300.));
   /* Create source app  */
   Address sinkAddress
(InetSocketAddress (i1i2.GetAddress(1),
sinkPort));
   Address sourceAddress
(InetSocketAddress (i0i1.GetAddress(0),
sinkPort));
   Ptr<Socket> socket =
Socket::CreateSocket (n.Get (0),
UdpSocketFactory::GetTypeId ());
```

```
   Ptr<QKDSend> app =
CreateObject<QKDSend> ();
   app->Setup (socket, sourceAddress,
sinkAddress, 640, 5, DataRate
("160kbps"));
   n.Get (0)->AddApplication (app);
   app->SetStartTime (Seconds (25.));
   app->SetStopTime (Seconds (300.));
   //if we need we can create pcap files
   p2p.EnablePcapAll
("QKD_channel_test");
Config::Connect("/NodeList/*/Applicatio
nList/*/$ns3::QKDSend/Tx",
MakeCallback(&SentPacket));

Config::Connect("/NodeList/*/Applicatio
nList/*/$ns3::QKDSink/Rx",
MakeCallback(&ReceivedPacket));

   AnimationInterface anim
("testcase1.xml");
   anim.SetConstantPosition(n.Get(0), 0.0,
200.0);
   anim.SetConstantPosition(n.Get(1),
200.0, 200.0);
   anim.SetConstantPosition(n.Get(2),
400.0, 200.0);

   Simulator::Stop (Seconds (50));
   Simulator::Run ();

   Ratio(app->sendDataStats(), app-
>sendPacketStats());

   //Finally print the graphs
   QHelper.PrintGraphs();
   Simulator::Destroy ();
}
```

3. Six node mesh network overlay implementation (sixmesh.cc)

```
#include <fstream>
#include "ns3/core-module.h"
#include "ns3/applications-module.h"
#include "ns3/internet-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/mobility-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/gnuplot.h"

#include "ns3/qkd-helper.h"
#include "ns3/qkd-app-charging-helper.h"
#include "ns3/qkd-send.h"

#include "ns3/aodv-module.h"
#include "ns3/olsr-module.h"
#include "ns3/dsdv-module.h"

#include "ns3/network-module.h"
#include "ns3/internet-apps-module.h"
#include "ns3/netanim-module.h"

#include <iostream>
#include <fstream>
#include <vector>
#include <string>

NS_LOG_COMPONENT_DEFINE
("SECOQC");

using namespace ns3;
uint32_t m_bytes_total = 0;
uint32_t m_bytes_received = 0;
uint32_t m_bytes_sent = 0;
uint32_t m_packets_received = 0;
double m_time = 0;

void
SentPacket(std::string context, Ptr<const
Packet> p){

   m_bytes_sent += p->GetSize();
}
void
ReceivedPacket(std::string context, Ptr<const
Packet> p, const Address& addr){

   m_bytes_received += p->GetSize();
   m_bytes_total += p->GetSize();
   m_packets_received++;
}
```

```
void
Ratio(uint32_t m_bytes_sent, uint32_t
m_packets_sent ){
   std::cout << "Sent (bytes):\t" <<
m_bytes_sent
   << "\tReceived (bytes):\t" <<
m_bytes_received
   << "\nSent (Packets):\t" <<
m_packets_sent
   << "\tReceived (Packets):\t" <<
m_packets_received

   << "\nRatio (bytes):\t" <<
(float)m_bytes_received/(float)m_bytes_sent
   << "\tRatio (packets):\t" <<
(float)m_packets_received/(float)m_packets_s
ent << "\n";
}
int main (int argc, char *argv[])
{
   Packet::EnablePrinting();
   PacketMetadata::Enable ();
   bool enableFlowMonitor = false;
   bool enableApplication = true;
   double simulationTime = 300;
   bool useSeparetedIPAddresses = false;
   std::string lat = "2ms";
   std::string rate = "10Mb/s"; // P2P link
   CommandLine cmd;
   cmd.AddValue ("EnableMonitor", "Enable
Flow Monitor", enableFlowMonitor);
   cmd.AddValue ("simulationTime",
"simulationTime", simulationTime);
   cmd.AddValue ("enableApplication",
"enableApplication", enableApplication);
   cmd.Parse (argc, argv);

   NS_LOG_INFO ("Create QKD system.");
   // Explicitly create the nodes required by the
topology (shown above).
   NS_LOG_INFO ("Create nodes.");
   NodeContainer nodes; // ALL Nodes
   nodes.Create(6);
   //Enable OLSR
   //OlsrHelper routingProtocol;
   //AodvHelper routingProtocol;
   DsdvHelper routingProtocol;
   // Install Internet Stack
```

```
  InternetStackHelper internet;
   internet.SetRoutingHelper
(routingProtocol);
   internet.Install (nodes);
   // Set up Addresses
   NS_LOG_INFO ("Create channels.");
   NS_LOG_INFO ("Assign IP Addresses.");
   Ipv4AddressHelper ipv4;
   // Explicitly create the channels required by
the topology (shown above).
   PointToPointHelper p2p;
   p2p.SetDeviceAttribute ("DataRate",
StringValue (rate));
   p2p.SetChannelAttribute ("Delay",
StringValue (lat));
   //Nodes 0 & 1
   NodeContainer link_0_1;
   link_0_1.Add(nodes.Get(0));
   link_0_1.Add(nodes.Get(1));
   NetDeviceContainer devices_0_1 =
p2p.Install (link_0_1);
   ipv4.SetBase ("10.1.1.0", "255.255.255.0");
   Ipv4InterfaceContainer ifconf_0_1 =
ipv4.Assign (devices_0_1);
   //Nodes 0 & 2
   NodeContainer link_0_2;
   link_0_2.Add(nodes.Get(0));
   link_0_2.Add(nodes.Get(2));
NetDeviceContainer devices_0_2 =
p2p.Install (link_0_2);

   if(useSeparetedIPAddresses)
      ipv4.SetBase ("10.1.2.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_0_2 =
ipv4.Assign (devices_0_2);
   //Nodes 0 & 3
   NodeContainer link_0_3;
   link_0_3.Add(nodes.Get(0));
   link_0_3.Add(nodes.Get(3));
   NetDeviceContainer devices_0_3 =
p2p.Install (link_0_3);
   if(useSeparetedIPAddresses)
      ipv4.SetBase ("10.1.3.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_0_3 =
ipv4.Assign (devices_0_3);
   //Nodes 0 & 4
   NodeContainer link_0_4;
   link_0_4.Add(nodes.Get(0));
   link_0_4.Add(nodes.Get(4));
```

```
NetDeviceContainer devices_0_4 =
p2p.Install (link_0_4);

   if(useSeparetedIPAddresses)
      ipv4.SetBase ("10.1.4.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_0_4 =
ipv4.Assign (devices_0_4);
   //Nodes 0 & 5
   NodeContainer link_0_5;
   link_0_5.Add(nodes.Get(0));
   link_0_5.Add(nodes.Get(5));

   NetDeviceContainer devices_0_5 =
p2p.Install (link_0_5);

   if(useSeparetedIPAddresses)
      ipv4.SetBase ("10.1.5.0",
"255.255.255.0");

   Ipv4InterfaceContainer ifconf_0_5 =
ipv4.Assign (devices_0_5);

   //Nodes 1 & 2
   NodeContainer link_1_2;
   link_1_2.Add(nodes.Get(1));
   link_1_2.Add(nodes.Get(2));

   NetDeviceContainer devices_1_2 =
p2p.Install (link_1_2);
   if(useSeparetedIPAddresses)
      ipv4.SetBase ("10.1.6.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_1_2 =
ipv4.Assign (devices_1_2);

//Nodes 1 & 3
   NodeContainer link_1_3;
   link_1_3.Add(nodes.Get(1));
   link_1_3.Add(nodes.Get(3));

   NetDeviceContainer devices_1_3 =
p2p.Install (link_1_3);
   ipv4.SetBase ("10.1.7.0", "255.255.255.0");
   Ipv4InterfaceContainer ifconf_1_3 =
ipv4.Assign (devices_1_3);

//Nodes 1 &4
   NodeContainer link_1_4;
   link_1_4.Add(nodes.Get(1));
   link_1_4.Add(nodes.Get(4));
```

```
NetDeviceContainer devices_1_4 =
p2p.Install (link_1_4);
   ipv4.SetBase ("10.1.8.0", "255.255.255.0");
   Ipv4InterfaceContainer ifconf_1_4 =
ipv4.Assign (devices_1_4);

//Nodes 1 & 5
   NodeContainer link_1_5;
   link_1_5.Add(nodes.Get(1));
   link_1_5.Add(nodes.Get(5));

   NetDeviceContainer devices_1_5 =
p2p.Install (link_1_5);
   ipv4.SetBase ("10.1.9.0", "255.255.255.0");
   Ipv4InterfaceContainer ifconf_1_5 =
ipv4.Assign (devices_1_5);

//Nodes 2 & 3
   NodeContainer link_2_3;
   link_2_3.Add(nodes.Get(2));
   link_2_3.Add(nodes.Get(3));

   NetDeviceContainer devices_2_3 =
p2p.Install (link_2_3);
   ipv4.SetBase ("10.1.10.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_2_3 =
ipv4.Assign (devices_2_3);

//Nodes 2 & 4
   NodeContainer link_2_4;
   link_2_4.Add(nodes.Get(2));
   link_2_4.Add(nodes.Get(4));
   NetDeviceContainer devices_2_4 =
p2p.Install (link_2_4);
   ipv4.SetBase ("10.1.11.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_2_4 =
ipv4.Assign (devices_2_4);

//Nodes 2 & 5
   NodeContainer link_2_5;
   link_2_5.Add(nodes.Get(2));
   link_2_5.Add(nodes.Get(5));

   NetDeviceContainer devices_2_5 =
p2p.Install (link_2_5);
   ipv4.SetBase ("10.1.12.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_2_5 =
ipv4.Assign (devices_2_5);
```

```
//Nodes 3 & 4
   NodeContainer link_3_4;
   link_3_4.Add(nodes.Get(3));
   link_3_4.Add(nodes.Get(4));

   NetDeviceContainer devices_3_4 =
p2p.Install (link_3_4);
   ipv4.SetBase ("10.1.13.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_3_4 =
ipv4.Assign (devices_3_4);

//Nodes 3 & 5
   NodeContainer link_3_5;
   link_3_5.Add(nodes.Get(3));
   link_3_5.Add(nodes.Get(5));

   NetDeviceContainer devices_3_5 =
p2p.Install (link_3_5);
   ipv4.SetBase ("10.1.14.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_3_5 =
ipv4.Assign (devices_3_5);

//Nodes 4 & 5
   NodeContainer link_4_5;
   link_4_5.Add(nodes.Get(4));
   link_4_5.Add(nodes.Get(5));

   NetDeviceContainer devices_4_5 =
p2p.Install (link_4_5);
   ipv4.SetBase ("10.1.15.0",
"255.255.255.0");
   Ipv4InterfaceContainer ifconf_4_5 =
ipv4.Assign (devices_4_5);



   // Create router nodes, initialize routing
database and set up the routing
   // tables in the nodes.

//Ipv4GlobalRoutingHelper::PopulateRouting
Tables ();
   //routingProtocol.Set
("LocationServiceName", StringValue
("GOD"));
   //routingProtocol.Install ();
   //  install QKD Managers on the nodes
   QKDHelper QHelper;
```

```
QHelper.InstallQKDManager (nodes);

   //create QKD connection between nodes 0
and 1
   NetDeviceContainer qkdNetDevices_0_1 =
QHelper.InstallQKD (
       devices_0_1.Get(0), devices_0_1.Get(1),
       1048576,    //min
       11324620, //thr
       52428800,   //max
       52428800     //current    //20485770
   );
   //Create graph to monitor buffer changes
   QHelper.AddGraph(nodes.Get(0),
devices_0_1.Get(0)); //srcNode,
destinationAddress, BufferTitle

   //create QKD connection between nodes 0
and 2
   NetDeviceContainer qkdNetDevices_0_2 =
QHelper.InstallQKD (
       devices_0_2.Get(0), devices_0_2.Get(1),
       1548576,    //min
       11324620, //thr
       52428800,   //max
       52428800     //current    //20485770
   );
   //Create graph to monitor buffer changes
   QHelper.AddGraph(nodes.Get(0),
devices_0_2.Get(1)); //srcNode,
destinationAddress, BufferTitle

//create QKD connection between nodes 0 and
3
   NetDeviceContainer qkdNetDevices_0_3 =
QHelper.InstallQKD (
       devices_0_3.Get(0), devices_0_3.Get(1),
       1048576,    //min
       11324620, //thr
       52428800,   //max
       10485760     //current    //10485760
);
   //Create graph to monitor buffer changes
   QHelper.AddGraph(nodes.Get(0),
devices_0_3.Get(1)); //srcNode,
destinationAddress, BufferTitle

//create QKD connection between nodes 0 and
4
   NetDeviceContainer qkdNetDevices_0_4 =
QHelper.InstallQKD (
devices_0_4.Get(0), devices_0_4.Get(1),
       1048576,    //min
       11324620, //thr
       52428800,   //max
       10485760     //current    //10485760
   );
   //Create graph to monitor buffer changes
   QHelper.AddGraph(nodes.Get(0),
devices_0_4.Get(1)); //srcNode,
destinationAddress, BufferTitle

//create QKD connection between nodes 0 and
5
   NetDeviceContainer qkdNetDevices_0_5 =
QHelper.InstallQKD (
       devices_0_5.Get(0), devices_0_5.Get(1),
       1048576,    //min
       11324620, //thr
       52428800,   //max
       10485760     //current    //10485760
   );
   //Create graph to monitor buffer changes
   QHelper.AddGraph(nodes.Get(0),
devices_0_5.Get(1)); //srcNode,
destinationAddress, BufferTitle

//create QKD connection between nodes 1 and
2
   NetDeviceContainer qkdNetDevices_1_2 =
QHelper.InstallQKD (
       devices_1_2.Get(0), devices_1_2.Get(1),
       1048576,    //min
       11324620, //thr
       52428800,   //max
       10485760     //current    //10485760
   );
   //Create graph to monitor buffer changes
   QHelper.AddGraph(nodes.Get(1),
devices_1_2.Get(1)); //srcNode,
destinationAddress, BufferTitle
//create QKD connection between nodes 1 and
3
   NetDeviceContainer qkdNetDevices_1_3 =
QHelper.InstallQKD (
       devices_1_3.Get(0), devices_1_3.Get(1),
       1048576,    //min
       11324620, //thr
       52428800,   //max
       10485760     //current    //10485760
);
   //Create graph to monitor buffer changes
```

```
QHelper.AddGraph(nodes.Get(1),
devices_1_3.Get(1)); //srcNode,
destinationAddress, BufferTitle

//create QKD connection between nodes 1 and
4
   NetDeviceContainer qkdNetDevices_1_4 =
QHelper.InstallQKD (
       devices_1_4.Get(0), devices_1_4.Get(1),
       1048576,    //min
       11324620, //thr
       52428800,   //max
       10485760     //current    //10485760
   );
   //Create graph to monitor buffer changes
   QHelper.AddGraph(nodes.Get(1),
devices_1_4.Get(1)); //srcNode,
destinationAddress, BufferTitle

//create QKD connection between nodes 1 and
5
   NetDeviceContainer qkdNetDevices_1_5 =
QHelper.InstallQKD (
       devices_1_5.Get(0), devices_1_5.Get(1),
       1048576,    //min
       11324620, //thr
       52428800,   //max
       10485760     //current    //10485760
   );
   //Create graph to monitor buffer changes
   QHelper.AddGraph(nodes.Get(1),
devices_1_5.Get(1)); //srcNode,
destinationAddress, BufferTitle

//create QKD connection between nodes 2 and
3
NetDeviceContainer qkdNetDevices_2_3 =
QHelper.InstallQKD (
       devices_2_3.Get(0), devices_2_3.Get(1),
       1048576,    //min
       11324620, //thr
       52428800,   //max
       10485760     //current    //10485760
   );
//Create graph to monitor buffer changes
   QHelper.AddGraph(nodes.Get(2),
devices_2_3.Get(1)); //srcNode,
destinationAddress, BufferTitle

   //create QKD connection between nodes 2
and 4
NetDeviceContainer qkdNetDevices_2_4 =
QHelper.InstallQKD (
       devices_2_4.Get(0), devices_2_4.Get(1),
       1048576,    //min
       11324620, //thr
       52428800,   //max
       10485760     //current    //10485760
   );
   //Create graph to monitor buffer changes
   QHelper.AddGraph(nodes.Get(2),
devices_2_4.Get(1)); //srcNode,
destinationAddress, BufferTitle

   //create QKD connection between nodes 2
and 5
   NetDeviceContainer qkdNetDevices_2_5 =
QHelper.InstallQKD (
       devices_2_5.Get(0), devices_2_5.Get(1),
       1048576,    //min
       11324620, //thr
       52428800,   //max
       52428800     //current    //20485770
   );
   //Create graph to monitor buffer changes
   QHelper.AddGraph(nodes.Get(2),
devices_2_5.Get(1)); //srcNode,
destihnationAddress, BufferTitle
   //create QKD connection between nodes 3
and 4
NetDeviceContainer qkdNetDevices_3_4 =
QHelper.InstallQKD (
       devices_3_4.Get(0), devices_3_4.Get(1),
       1048576,    //min
       11324620, //thr
       52428800,   //max
       12485760     //current    //12485760
   );
   //Create graph to monitor buffer changes
   QHelper.AddGraph(nodes.Get(3),
devices_3_4.Get(1)); //srcNode,
destinationAddress, BufferTitle
//create QKD connection between nodes 3 and
5
   NetDeviceContainer qkdNetDevices_3_5 =
QHelper.InstallQKD (
       devices_3_5.Get(0), devices_3_5.Get(1),
       1048576,    //min
       11324620, //thr
       52428800,   //max
       52428800     //current    //20485770
   );
```

```
//QKD LINK 1_2
   QKDAppChargingHelper
qkdChargingApp_1_2("ns3::TcpSocketFactor
y", ifconf_1_2.GetAddress(0),
ifconf_1_2.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer qkdChrgApps_1_2 =
qkdChargingApp_1_2.Install (
devices_1_2.Get(0), devices_1_2.Get(1) );
   qkdChrgApps_1_2.Start (Seconds (5.));
   qkdChrgApps_1_2.Stop (Seconds (500.));
   //QKD LINK 1_3
   QKDAppChargingHelper
qkdChargingApp_1_3("ns3::TcpSocketFactor
y", ifconf_1_3.GetAddress(0),
ifconf_1_3.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer qkdChrgApps_1_3 =
qkdChargingApp_1_3.Install (
devices_1_3.Get(0), devices_1_3.Get(1) );
   qkdChrgApps_1_3.Start (Seconds (5.));
   qkdChrgApps_1_3.Stop (Seconds (500.));
   //QKD LINK 1_4
   QKDAppChargingHelper
qkdChargingApp_1_4("ns3::TcpSocketFactor
y", ifconf_1_4.GetAddress(0),
ifconf_1_4.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer qkdChrgApps_1_4 =
qkdChargingApp_1_4.Install (
devices_1_4.Get(0), devices_1_4.Get(1) );
   qkdChrgApps_1_4.Start (Seconds (5.));
   qkdChrgApps_1_4.Stop (Seconds (500.));
   //QKD LINK 1_5
   QKDAppChargingHelper
qkdChargingApp_1_5("ns3::TcpSocketFactor
y", ifconf_1_5.GetAddress(0),
ifconf_1_5.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer qkdChrgApps_1_5 =
qkdChargingApp_1_5.Install (
devices_1_5.Get(0), devices_1_5.Get(1) );
   qkdChrgApps_1_5.Start (Seconds (5.));
   qkdChrgApps_1_5.Stop (Seconds (500.));

   //QKD LINK 2_3
   QKDAppChargingHelper
qkdChargingApp_2_3("ns3::TcpSocketFactor
y", ifconf_2_3.GetAddress(0),
ifconf_2_3.GetAddress(1), 3072000);
//102400 * 30seconds

   qkdChrgApps_2_3.Stop (Seconds (500.));
```

```
ApplicationContainer qkdChrgApps_2_3 =
qkdChargingApp_2_3.Install (
devices_2_3.Get(0), devices_2_3.Get(1) );
   qkdChrgApps_2_3.Start (Seconds (5.));
   qkdChrgApps_2_3.Stop (Seconds (500.));
   //QKD LINK 2_4
   QKDAppChargingHelper
qkdChargingApp_2_4("ns3::TcpSocketFactor
y", ifconf_2_4.GetAddress(0),
ifconf_2_4.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer qkdChrgApps_2_4 =
qkdChargingApp_2_4.Install (
devices_2_4.Get(0), devices_2_4.Get(1) );
   qkdChrgApps_2_4.Start (Seconds (5.));
   qkdChrgApps_2_4.Stop (Seconds (500.));
   //QKD LINK 2_5
   QKDAppChargingHelper
qkdChargingApp_2_5("ns3::TcpSocketFactor
y", ifconf_2_5.GetAddress(0),
ifconf_2_5.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer qkdChrgApps_2_5 =
qkdChargingApp_2_5.Install (
devices_2_5.Get(0), devices_2_5.Get(1) );
   qkdChrgApps_2_5.Start (Seconds (5.));
   qkdChrgApps_2_5.Stop (Seconds (500.));

   //QKD LINK 3_4
   QKDAppChargingHelper
qkdChargingApp_3_4("ns3::TcpSocketFactor
y", ifconf_3_4.GetAddress(0),
ifconf_3_4.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer qkdChrgApps_3_4 =
qkdChargingApp_3_4.Install (
devices_3_4.Get(0), devices_3_4.Get(1) );
   qkdChrgApps_3_4.Start (Seconds (5.));
   qkdChrgApps_3_4.Stop (Seconds (500.));

   //QKD LINK 3_5
   QKDAppChargingHelper
qkdChargingApp_3_5("ns3::TcpSocketFactor
y", ifconf_3_5.GetAddress(0),
ifconf_3_5.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer qkdChrgApps_3_5 =
qkdChargingApp_3_5.Install (
devices_3_5.Get(0), devices_3_5.Get(1) );
   qkdChrgApps_3_5.Start (Seconds (5.));
   qkdChrgApps_3_5.Stop (Seconds (500.));
```

```
//QKD LINK 4_5
   QKDAppChargingHelper
qkdChargingApp_4_5("ns3::TcpSocketFactor
y", ifconf_4_5.GetAddress(0),
ifconf_4_5.GetAddress(1), 3072000);
//102400 * 30seconds
   ApplicationContainer qkdChrgApps_4_5 =
qkdChargingApp_4_5.Install (
devices_4_5.Get(0), devices_4_5.Get(1) );
   qkdChrgApps_4_5.Start (Seconds (5.));
   qkdChrgApps_4_5.Stop (Seconds (500.));



   Ptr<QKDSend> app =
CreateObject<QKDSend> ();
   if(enableApplication){
      /* Create user's traffic between v0 and v1
*/
      /* Create sink app */
      uint16_t sinkPort = 8080;
      QKDSinkAppHelper packetSinkHelper
("ns3::UdpSocketFactory", InetSocketAddress
(Ipv4Address::GetAny (), sinkPort));
      ApplicationContainer sinkApps =
packetSinkHelper.Install (nodes.Get (5));
      sinkApps.Start (Seconds (15.));
      sinkApps.Stop (Seconds (500.));

      /* Create source app */
      Address sinkAddress (InetSocketAddress
(ifconf_0_5.GetAddress(1), sinkPort));
      Address sourceAddress
(InetSocketAddress
(ifconf_0_1.GetAddress(0), sinkPort));
      Ptr<Socket> socket =
Socket::CreateSocket (nodes.Get (0),
UdpSocketFactory::GetTypeId ());
app->Setup (socket, sourceAddress,
sinkAddress, 512, 0, DataRate ("160kbps"));
      nodes.Get (0)->AddApplication (app);
      app->SetStartTime (Seconds (15.));
      app->SetStopTime (Seconds (500.));
   }
//if we need we can create pcap files
   p2p.EnablePcapAll ("QKD_netsim_test");
   //QHelper.EnablePcapAll
("QKD_netsim_test_Qhelper");

Config::Connect("/NodeList/*/ApplicationList
/*/$ns3::QKDSink/Rx",
MakeCallback(&ReceivedPacket));
```

```
   //
   // Now, do the actual simulation.
   //
   NS_LOG_INFO ("Run Simulation.");

   //AnimationInterface
anim("secoqc_olsr_wifi.xml");

AnimationInterface anim
("animsixmesh.xml");
  anim.SetConstantPosition(nodes.Get(0), 0.0,
200.0, 0.0);
  anim.SetConstantPosition(nodes.Get(1),
100.0, 0.0, 0.0);
  anim.SetConstantPosition(nodes.Get(2),
300.0, 0.0, 0.0);
  anim.SetConstantPosition(nodes.Get(3),
500.0, 200.0, 0.0);
  anim.SetConstantPosition(nodes.Get(4),
300.0, 400.0, 0.0);
  anim.SetConstantPosition(nodes.Get(5),
100.0, 400.0, 0.0);
   Simulator::Stop
(Seconds(simulationTime));
   Simulator::Run ();

   if(enableApplication)
      Ratio(app->sendDataStats(), app-
>sendPacketStats());

   //Finally print the graphs
   QHelper.PrintGraphs();
   Simulator::Destroy ();
}
```