

Random State Production for Quantum Key Distribution using Weak Coherent Pulse Source

by

Matthew Piatt

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Science
in
Physics (Quantum Information)

Waterloo, Ontario, Canada, 2024

© Matthew Piatt 2024

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis contains work done in collaboration with others but to which I made the major contribution. The contributions are listed below.

Chapter 1

Matthew Piatt wrote the introduction

Chapter 2

Thomas Jennewein and Brendon Higgins provided background on the QRNG scheme

Joe Itoi wrote the initial version of the QRNG processing software

Matthew Piatt corrected initial version, re implemented the software

Matthew Piatt incorporated the t -digest into the QRNG software

Matthew Piatt incorporated the Toeplitz extractor into the QRNG software

Simon Carrier provided the physical system for implementing the QRNG

Chapter 3

Ramy Tannous, Thomas Jennewein, and Brendon Higgins provided background on the AWG

Ramy Tannous, Thomas Jennewein, and Brendon Higgins selected and purchased the AWG

Matthew Piatt developed the streaming software

Abstract

This thesis addresses two challenges involved in the culmination of the Quantum Encryption and Science Satellite mission. This mission aims to demonstrate quantum key distribution (QKD) in space. The first part of the thesis will be dedicated to the introduction of a quantum random number generator. This generator will be based off of the arrival time of photons and subsequently used to produce random bits that will be turned into the random states necessary for the QKD protocol. The latter half of the thesis will address creating the random states for the protocol. This is non-trivial due to the finite resources involved in the physical apparatus that comprises the chosen weak coherent pulse source for the QEYSSat mission.

Acknowledgments

First, I'd like to thank my supervisor, Thomas Jennewein, for his patience and understanding. I also would like to especially thank Brendon Higgins for his guidance and support, without which I likely would not be writing this thesis. Next, I'd like to thank my committee members, Crystal Senko and Michael Reimer, for their advice and direction.

I want to take a moment and thank my colleagues and friends in QPL and IQC who made each day a little brighter. I cannot express how appreciative I am of all of you. Special thanks to Steve Weiss for his never ending supply of milk chocolate and words of wisdom. The frequent trips downstairs for discussion and sugar were always worthwhile.

Lastly, I want to acknowledge the Natural Sciences and Engineering Research Council of Canada (NSERC) as the primary funding agency that supported me during my masters.

Table of Contents

Author's Declaration	ii
Statement of Contributions	iii
Abstract	iv
Acknowledgments	v
List of Figures	ix
List of Tables	xi
1 Introduction	1
1.1 Thesis Overview	1
1.2 Statistical Background	2
1.3 Background	4
1.4 Quantum Key Distribution	7
1.4.1 BB84 Protocol	8
1.5 Randomness	11
1.5.1 Entropy	12
1.5.2 Quantum Key Distribtuion and Weak Coherent Pulse sources . . .	15

2	Arrival Time Quantum Random Number Generator	18
2.1	QEYSSat Requirement	18
2.2	"Time of Arrival" QRNG	21
2.3	Implementation	25
2.3.1	Post Processing algorithm	25
2.3.2	Initial Results	28
2.4	Implementing Randomness Extractor	34
2.4.1	Min Entropy	34
2.4.2	Toeplitz Extractor	35
2.5	Results	38
2.5.1	Characterization Curve and Quantile Function	44
2.6	Outlook	44
2.6.1	Improvements	47
3	High-Speed Streaming Software	50
3.1	QEYSSat Requirement	50
3.2	Arbitrary Waveform Generator Overview	51
3.2.1	Arbitrary Waveform Generator	51
3.3	Streaming Overview	53
3.3.1	QEYSSat WCPS	53
3.3.2	Streaming File Format	57
3.4	Streaming Software Implementation	59
3.4.1	Streaming Software Iteration 1	59
3.4.2	Streaming Software Iteration 2	63
3.5	Outlook	73
4	Conclusion and Outlook	76
	References	78

APPENDICES	85
.1 SPAD Camera	85
.2 Characteristic Curve	87

List of Figures

1.1 Rectilinear and diagonal axis bases.	9
1.2 Airborne WCPS	17
2.1 Example Algorithm for choosing Intensity	20
2.2 Simulated Bin Distributions	24
2.3 QRNG post processing scheme	27
2.4 NIST STS Initial Results	28
2.5 SPAD array heat map of counts	29
2.6 Initial Bin Distribution For Select Pixels	30
2.7 Characterization Curve Convergence Test	31
2.8 Bin Distribution using t -digest	33
2.9 The entire post processing scheme for the QRNG.	39
2.10 Typical min entropy results	40
2.11 Toeplitz Extractor Impact	42
2.12 Characterization Curve Versus Quantile Function	45
2.13 SOA schematic	49
2.14 TOA and SOA parallel processing	49
3.1 High level AWG operation	52
3.2 Tabor P2584M model AWG	53
3.3 WCPS block diagram	54

3.4	Interleaved versus disjoint file structure	56
3.5	WCP to AWG samples program flow chart	58
3.6	Streaming Software iteration 1 flow chart	60
3.7	Streaming software iteration 1 performance results	61
3.8	Crystal Disk Performance Software Result	63
3.9	Synchronous IO	66
3.10	Asynchronous IO	67
3.11	Streaming software iteration 2 flow chart	69
3.12	Streaming software iteration 2 performance results	70
3.13	Producer Consumer implementation alternative metaphor	72
3.14	Streaming Software output example	73
1	Camera Dark Count Rate	86

List of Tables

2.1	NIST STS results after processing raw bits with Toeplitz extractor	43
3.1	Post Thread Consolidation performance of streaming software	73
1	Characterization Curve Values for Pixel 31	87

Chapter 1

Introduction

Quantum information is one of many applications of quantum theory. While the improved understanding of the physical world has led to more tangible results like the invention of magnetic resonance imaging (MRI) in 1974 or the transistor in 1947. More recently, the continued development of quantum computers has brought the implications of Peter Shor's 1994 paper that introduced the world to a faster factoring algorithm that takes advantage of quantum theory. Modern encryption methods like RSA that are widely used to protect confidential information are essentially based on the assumed hardness of factoring large numbers. Shor's algorithm invalidates that assumption provided a fully functioning quantum computer is available to implement the algorithm. This has reignited interest into being able to assuredly secure information against a quantum adversary. The study of improving the storage and transmission of information with quantum enhancements encompasses the field of quantum information.

1.1 Thesis Overview

The Quantum EncrYption and Science Satellite (QEYSSat) mission's goal is to demonstrate the usage of satellite based Quantum Key Distribution (QKD) to establish a secure communication link between two ground stations[29]. One of the quantum sources used in the mission is a weak coherent pulse source (WCPS) in order to perform the decoy-state BB84 protocol[27]. With any protocol there is a need to produce random states. The work presented in this thesis is concerned with the production of random states by developing a Quantum Random Number Generator (QRNG) and subsequently transforming them into weak coherent pulse states via the WCPS used in the decoy state protocol.

There are many different methods concerned with extracting random bits from a quantum system. The earliest QRNG's were based off of radioactive decay[25]. This setup takes advantage of the probability for an atom to decay being distributed exponentially to obtain random bits. However, the need for faster generation rates pushed development away from highly radioactive sources. From there efforts came to be focused on photonic systems. The arrival time of photons will be used as the basis for the QRNG to be described in a later chapter. Similarly, for WCPS's the methods of transforming random data into random states can vary. In this work a streaming algorithm is presented to interface with driving electronics of the WCPS to facilitate random state production.

The rest of this thesis is structured as follows. The remainder of this chapter serves to give a background on the concepts related to the work to be presented. Starting with a brief statistical background for a couple of key probability distributions. Next is an overview of photons and their number distribution. From there a discussion of the original BB84 protocol as it is closely related to its decoy state counter part. Following this is a discussion on randomness, random number generators, entropy, and verification of random bits. Lastly, a brief discussion on weak coherent pulse sources and quantum key distribution as the usage of a weak coherent pulse source led to this thesis.

The second chapter is focused on the QRNG. Initially detailing the algorithm employed to produce random bits and its shortcomings. After which subsequent changes and additions to the post processing scheme are examined. Lastly, the current status of the algorithm is described along with avenues for potential improvements. Additionally, a short term method for integration into the QEYSSat mission is also presented.

The third chapter concerns the development of the streaming software and how it fits into the WCPS currently planned as one of the two used in the QEYSSat mission. Characteristics related to the key electrical device that is used, an arbitrary waveform generator, are examined. This streaming software is inherently less related to the inner working of the WCPS and more concerned with concepts relevant to data transmission. To this end miscellaneous software scripts related to transforming random bits to random WCP states are examined in conjunction with modalities of streaming this data efficiently.

The fourth chapter concludes this thesis and briefly reviews the avenues for future work.

1.2 Statistical Background

While an in-depth statistical background is unnecessary for the content of this thesis, there are a few relevant statistical functions and concepts that will be introduced. The first is

the Probability Mass Function (PMF) used in the description of discrete random variables. The PMF of a discrete random variable X is

$$p(x) = P(X = x) \quad (1.1)$$

returning the various probabilities for the possible values X can have. Related to this is the Cumulative Distribution Function (CDF). The CDF for a particular discrete random variable X is defined as $F(x) = P(X \leq x)$ and yields the probability that X will take on a value less than or equal to x . This is closely related to the idea of a quantile, q , which can be used to partition a distribution into potentially equiprobable slices¹. For a strictly increasing and continuous CDF, the quantile function $F^{-1}(p), p \in [0, 1]$ will return a value x such that $F(x) = p$.

Poissonian Distribution

Of particular importance to the content of this thesis is the Poisson distribution which arises when describing photon number statistics, provided the observed photons events are random and independent of each other. The PMF of the Poissonian distribution is

$$f(k : \lambda) = P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!} \quad (1.2)$$

and it commonly arises when counting occurrences of events[18]. A closely related distribution arises when attempting to describe the interval between consecutive events. This is known as the waiting time distribution and follows the exponential distribution which has a probability density function (continuous analog to PMF) of

$$\lambda e^{-\lambda x} \quad (1.3)$$

where x is the probability of observing an event in a particular interval. λ is the main parameter of the distribution, known as the rate parameter. This distribution is continuous, unlike the discrete distribution it is derived from². Its CDF is determined by integrating the probability density function

$$\int_{0 \rightarrow x} \lambda e^{-\lambda x} dx = 1 - e^{-\lambda x} \quad (1.4)$$

¹In practice, the quantile is a measure of event frequency that is dependant on how representative the data sample being analyzed is of the population it was sampled from

²Specifically, the exponential distribution comes from the waiting time between continuous and independent events associated with a Poissonian point process

and its quantile function is

$$F^{-1}(p, \lambda) = -\frac{\ln 1 - p}{\lambda} \quad (1.5)$$

One important property of the exponential distribution is that it is memoryless[18]. This property is normally associated with waiting times, and its meaning is that the waiting time is independent of the time already waited. As an example, take a situation where an individual was attempting to guess the password of a locked computer, and there were x identical locked computers. It would be expected to take $\frac{1}{x}$ attempts to guess the password of one of these identical computers, even if the individual had already guess $x-1$ times on $x-1$ different computers. This is due to the fact that the individual retains no knowledge of the password that was just guessed. Mathematically, for two real numbers t and s for a continuous random variable X

$$P(X > t + s | X > s) = P(x > s) \quad (1.6)$$

One example of a real-world phenomena that exhibits this exponential distribution is radioactive decay[18].

1.3 Background

A common method of transmitting quantum information involves photons. The concept of a photon originally arose after the classical light wave previously described by Maxwell's equations was quantized. It turns out that these elemental quanta of the classical wave can be encompassed by the study of the quantum harmonic oscillator. The oscillator has a Hamiltonian

$$\hat{\mathcal{H}} = \frac{1}{2} \left(\frac{\hat{p}^2}{m} + m\omega^2 \hat{x}^2 \right) \quad (1.7)$$

Where $\hat{p}$ and $\hat{x}$ refer to momentum and position operators respectively. This Hamiltonian can be solved algebraically with the introduction of the creation and annihilation operators[58].

$$\hat{a} = \sqrt{\frac{m\omega}{2\hbar}} \left(\hat{x} + \frac{i\hat{p}}{m\omega} \right) \quad (1.8)$$

$$\hat{a}^\dagger = \sqrt{\frac{m\omega}{2\hbar}} \left(\hat{x} - \frac{i\hat{p}}{m\omega} \right) \quad (1.9)$$

subject to the usual commutation relation $[\hat{a}, a^\dagger] = 1$. With this the above Hamiltonian can be rewritten as

$$\hat{\mathcal{H}} = \hbar\omega(a^\dagger\hat{a} + \frac{1}{2}) \quad (1.10)$$

[58] and with the introduction of the number operator $\hat{n} := a^\dagger\hat{a}$ this Hamiltonian's spectrum can be determined in terms of the number states $|n\rangle$ subject to the eigenvalue equation $\hat{n}|n\rangle = n|n\rangle$. These number states are also referred to as the Fock states, and the collection of these number states form a complete orthonormal basis, termed the Fock basis[58].

Of particular relevance to photons is the coherent state, $|\alpha\rangle$, with $\alpha = |\alpha|e^{i\theta}$. The coherent state turns out to be a good approximation of the output of a single mode laser[37]. It is also an eigenstate of the annihilation operator[58]

$$a^\dagger |\alpha\rangle = \alpha |\alpha\rangle \quad (1.11)$$

with an important recurrence relation

$$\sqrt{n+1} \langle n+1|\alpha\rangle = \alpha \langle n|\alpha\rangle \quad (1.12)$$

That allows the expansion of the coherent state in the Fock basis

$$|\alpha\rangle = e^{-\frac{|\alpha|^2}{2}} \sum_{n=0}^{\infty} \frac{\alpha^n}{\sqrt{n!}} |n\rangle \quad (1.13)$$

Which allows the determination of the photon number distribution, $P(n)$, for the coherent state

$$P(n) = |\langle n|\alpha\rangle|^2 = \frac{|\alpha|^{2n}}{n!} e^{-|\alpha|^2} \quad (1.14)$$

Equation 1.14 is nothing more than the Poissonian PMF with parameter $|\alpha|^2$, the mean photon number. This means, depending on the mean photon number 0, 1, or more than 1 photons can be observed in a particular interval t.

Detecting Photons

In order to observe a photon, a single photon detector (SPD) is used. There are many different kinds of SPD's, one of which is the single photon avalanche diode or SPAD. The SPAD is based off of a P-N junction, and has two distinct modes of operation. The first is a linear mode, where the incoming optical signal causes a linear output response of the detector. In the second, the so called Geiger mode, the gain of the device is exponential

depending on its bias. This allows the device to become sensitive to a single photon, and the resultant avalanche of electrons in response to this single photon gives the device their name. For the device to detect subsequent photons, the avalanche must be quenched, and during this time the SPAD will not detect subsequent photons. This is referred to as the devices dead time. Related to the act of quenching, sometimes electrons remain trapped and are later released. This can cause spurious correlated counts referred to as afterpulses. Additionally, detectors exhibit dark counts which are associated with thermal noise causing a click to occur in the absence of photons. The rate of these dark counts is referred to as the dark count rate (DCR) Lastly, the detectors themselves have a range of sensitivity dependent on the wavelength of the incoming photon. This photon detection efficiency (PDE) characterizes how reliably the detector will count a single photon at a specific wavelength. A good review of detectors and their properties can be found in ref [23]

Quantum mechanically, the act of detecting photons is associated with a specific kind of operator, a 'positive operator-valued measure' or POVM operator. These operators can be intuitively thought of as the most general measurement one can do on a system. The POVM associated with photon detection, assuming a perfect photon number resolving detector³, is the projection onto the corresponding number state. Following the notation by Kok and and Lovett this operator is expressed as [35]

$$\hat{P}^n = |n\rangle \langle n| \quad (1.15)$$

which is then the POVM associated with detecting a particular number state $|n\rangle$. In the case of a non-ideal or bucket detector the associated POVM must cover all number states or none of them. In simpler terms, the detector either clicks or does not click. These two POVM's are then

$$\hat{P}^{(click)} = \sum_{n=1}^{\infty} \hat{P}^n = \sum_{n=1}^{\infty} |n\rangle \langle n| \quad \hat{P}^{(no-click)} = \hat{\mathbb{1}} - \hat{P}^{click} = \hat{P}^0 = |0\rangle \langle 0| \quad (1.16)$$

Where the probability of click and no-click sum to one. Now, the probability of detecting a photon from a state ρ is then determined by the trace of $\hat{P}^{(click)}\rho$ which is equal to

$$\sum_{n=1}^{\infty} \langle n|\rho|n\rangle \quad (1.17)$$

³A photon number resolving detector can differentiate between 0, 1,2,...,n photons hitting the detector. A detector that can only differentiate between 0 and at least 1 photon is referred to as a bucket detector.

It should be noted that this is the case for an ideal detector with perfect efficiency. A more realistic model can be introduced to accommodate finite-efficiency detectors, where the detectors efficiency λ can be modeled as having a beam splitter with transmittance equal to λ . Additionally, dark counts can be accounted for without making assumptions related to the actual probability distribution for the dark counts. For more details, refer to section 4.1 of reference [35].

1.4 Quantum Key Distribution

Classically, key distribution is done with a public and private key pair. The security of the public key cannot be guaranteed unless the key length is the same as the message, such as the case with a one-time pad[68]. As the name suggests, the largest downside of the one-time pad is that it can only be used once, followed closely by the fact that larger messages imply larger key lengths. Therefore, classical key distribution is rarely done with the assistance of a one-time pad. Quantum key distribution (QKD) seeks to fill the gap between mostly secure and completely secure communication by guaranteeing security not through perceived computational hardness but by the laws of quantum mechanics. The two fundamental laws that assure complete security are the Heisenberg uncertainty principle[24] and the No Cloning-Theorem[71].

A common version of the uncertainty principle is expressed below

$$\delta_x \delta_p \geq \frac{\hbar}{2} \quad (1.18)$$

Which translates to the product of the uncertainties in position and momentum must always be greater than or equal to a fundamental constant. This implies that increased precision in position corresponds to decreased precision in momentum and vice versa. Position and momentum are referred to as complementary variables in the sense that these two properties of a system cannot be observed simultaneously. Therefore, if a quantum system transmitted information using complementary variables, such as polarization along two axes separated by 45° [72][55], then the uncertainty principle would limit the amount of information able to be obtained by interacting with the system.

The no cloning theorem is straightforward, a quantum state cannot be copied. This is in contrast to classically being able to copy a bit of information with ease without any regard for what the information actually is. Combined with the uncertainty principle, a recipe is obtained for securing communications in a way that cannot be done classically. This was first stated by Bennet and Brassard in what is now known as the first QKD protocol, BB84[7].

1.4.1 BB84 Protocol

Following the convention set in the original paper, the two parties seeking to communicate securely will be referred to as Alice and Bob. They will be interacting with a quantum state, namely the polarization of photons. These photons will be sent in either one of two polarization bases. The first is the horizontal and vertical polarization basis or Z basis. The second is the diagonal and anti-diagonal basis or X basis. Each photon that is sent contains one bit of information, meaning that each photon is a quantum bit or qubit of information. Now the entire protocol can be sketched out in 5 steps

1. State preparation
2. State transmission
3. State reception and measurement
4. Reconciliation
5. Privacy amplification

Before elaborating on each of these steps I will first give a brief introduction into using photons as qubits.

Qubits

Classically, a bit can be thought of as a boolean variable. To check the state of the bit, one can interrogate the system for the current value of that boolean. There is no ambiguity about the state of the classical bit. This is not the case for a qubit, where an analogy of a boolean variable is insufficient. Rather, the quantum system that is being used as a qubit is not so simple. Take the stereotypical two-level system that is ubiquitous throughout quantum theory[58]. The two states of that system are linearly independent of each other, therefore they form a basis to perform computation with. If one were to think of the Cartesian plane, one state forms the x-axis and the other the y-axis. All the possibilities between x and y are a combination of the two states or more succinctly a superposition of them. Most two level systems are good candidates for quantum bits, and one such example is the polarization of a photon. One can take two orthogonal polarization's, such as horizontal and vertical, and use them as two independent states to form a basis with. Normally, the horizontal polarization, $|H\rangle$ is chosen to be the $|0\rangle$ state and the vertical polarization, $|V\rangle$ is chosen to be the $|1\rangle$ state. This basis is often referred to as the

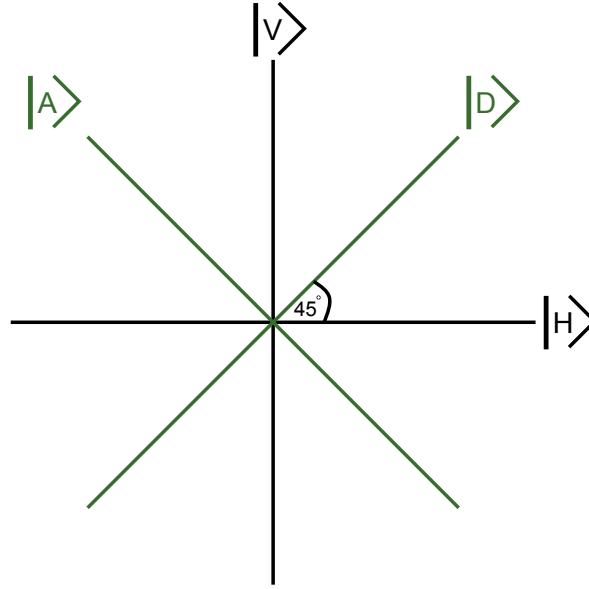


Figure 1.1: Rectilinear and diagonal axis bases.

rectilinear basis, but was previously introduced as the Z-basis. A complementary basis to the Z-basis is the diagonal or x-basis, where the diagonal, $|D\rangle$, state is often considered $|0\rangle$ and the anti-diagonal, $|A\rangle$, state is usually $|1\rangle$. The two pairs of polarization states $(|H\rangle, |V\rangle)$, and $(|D\rangle, |A\rangle)$ are complimentary variables. A visual representation of the two bases is given in [Figure 1.1](#)

One thing to note is that polarization in it of itself is not enough to be considered a qubit[66]. The polarization of a classical wave can be measured without changing its state. On the other hand, a photons polarization cannot be measured without disturbing the state of the photon. Therefore, it is the combination of the photon with a polarization that can be used as a qubit. With this in mind the BB84 protocol can be understood more intuitively.

State Preparation

Recall in the original setup there are two bases, the Z and X basis. Initially, Alice and Bob will determine which of these bases to use as the computational basis. Next, Alice will randomly select a basis and polarization state to send to bob. She will keep track of the selections for use later in the protocol. The act of randomly selecting a basis and polarization state is done with the help of an entropy source that can be turned into random

states. One example is the use of a random number generator whose output is used to generate random states. Next the physical state must be prepared, and this depends on the quantum source used to generate these random states. A common choice is the use of a weak coherent pulse source, which was mentioned earlier in this chapter. While a WCPS only emulates a single photon source, it can still be used for QKD and this is described a bit more in [subsection 1.5.2](#).

State Transmission

After preparing the states the next step is for Alice to transmit the states to Bob. Commonly, the signal was sent via an optical fiber at a specified wavelength. This is analogous to the modern telecommunication setup widely employed worldwide. One downside of using an optical fiber is that the losses experienced by the signal are quite severe[73]. For classical communications, this is overcome with classical repeaters or signal boosters. Naively, these can be thought of as copying the original state and re-sending it to reach longer distances. Due to the no-cloning theorem, this solution does not apply when sending a quantum signal and research into quantum repeaters is an active field of study. Instead, quantum repeaters are being developed using quantum entanglement and teleportation processes in order to transfer the states further rather than duplicate them[59].

State Reception and Measurement

Now, as Bob receives the states he will randomly select a basis to measure the state and record the result. There are some subtleties here related to the detectors that can undermine the security of the protocol, one example being detector efficiency mismatches that can be used to compromise the system. However, as far as the protocol is concerned, that's all there is to this step.

Basis sifting

Both parties now communicate over a classical channel to compare the basis Alice prepared with the basis Bob measured, and when the two agree the result will be kept. Otherwise they will discard the bit that corresponds to that measurement. Repeating this process over all transmitted bits allows Alice and Bob to generate a raw key. With the assumption of no eavesdropping (amongst others) the protocol would stop here and the raw key would become the secret key. However, the validity of this sifted key is determined by comparing

a subset of the original measured results from Bob with the corresponding subset of states Alice sent. If enough of the bits agree, this sifted key is kept. Otherwise, the key is discarded and the protocol aborted.

Classical Post Processing

The last step in the protocol deals with refining the key further to address non-ideal aspects of the previous steps. Assuming no eavesdropping and a perfect physical implementation is not realistic. Therefore, instead one attempts to minimize the amount of information a potential eavesdropper can obtain undetected. This is typically broken up into two generic stages. The first is an error correction stage to remove differences between Alice and Bob's sifted key. Next is privacy amplification using a universal hashing algorithm. After each stage the initial sifted key will be reduced further. The final secret key is then private, with little to no information leakage to an outsider.

1.5 Randomness

Defining randomness is a difficult task, so much so that mathematicians typically avoid directly addressing the subject. A physical definition of randomness continues to elude researchers to this day, with a widespread acceptance that defining randomness is exceptionally difficult if not impossible. In most publications, one notes that the usual method is to discuss the properties of a sequence of random numbers and the uniform distribution they arise from[63]. The actual methods involved in producing these random numbers - or sampling this distribution - are left to the reader to decide upon, which makes designing a random number generator that abides by the theory rather problematic. Ultimately, researchers agree on parameters that random numbers should abide by, and this can be seen through the use of rigorous randomness verification libraries such as NIST STS[5]. Regardless, the development of random number generators has led to two main distinct classes of generators. These two classes are psuedo-random number generators (PRNG's) and true random number generators (TRNG's)[25].

PRNG's are algorithmic and ultimately deterministic. They seek to emulate the behavior of a random source by producing an output bit sequence that is not random but appears to be so[34]. PRNG's have a periodicity and are based off of a seed. If the periodicity of the PRNG is eclipsed, meaning that an output bit sequence larger than its period is requested, the pattern can be identified and the masquerade of randomness dissipates. Similarly, PRNG's are seeded, where the seed is some sequence of random or non-random

bits that determine the output. If the seed is known, the output of the PRNG is known. In contrast to this true random number generators seek to extract 'true' random bits from a physical system with high entropy. This in turn leads to a related but slightly different class of generator termed a physical random number generator. This generic class is further differentiated by the type of physical system being used to extract random bits, an example being hardware random number generators. A personal favorite example of a physical random number generator is the lava lamp[52]. The actual state of the lava lamp at any one point in time is difficult to predict, and by extension this unpredictability can be exploited to generate true random bits. The caveat for these kind of physical random number generators is that even a chaotic systems evolution can be inferred if its initial conditions are known. This has the potential to return a physical random number generator to the same status of a PRNG whose seed is known. In contrast to this, quantum random number generators (QRNG) are physical random number generators that interact with a quantum system to extract random bits. Here, the laws of quantum mechanics apply and there is always a degree of unpredictability in the exact state of the system. In other words, gaining information about the system does not allow us to predict with absolute certainty the result of a measurement. It is this aspect of QRNG's that drives interest in developing this type of physical RNG[25]. That being said, it should be made clear that any type of physical RNG can be considered a TRNG regardless if it uses a classical or quantum system. The key difference between the myriad types of physical RNG's lies in how difficult it is to undermine the underlying process for true random bit generation.

1.5.1 Entropy

In the previous introduction to random number generators the concept of entropy was used but not introduced. Entropy has different definitions that arise in the context to which it is being applied. For the work to be presented in this thesis the definition of entropy from information theory is taken. Meaning entropy can be intuitively thought of as the degree of uncertainty present in a system. This is normally associated with Shannon's entropy,

$$H(X) := - \sum_{x \in \chi} p_x \log_2(p_x) \quad (1.19)$$

where χ is some distribution and $p(x)$ is the probability of observing x from the distribution χ . X is a discrete random variable, which can take on values dependent on χ , x , and $p(x)$. Shannon's entropy is typically measured in bits. To give a simple example of applying Shannon's entropy consider tossing a fair coin. There are only two possible outcomes and

they each have the same probability, 0.5. Therefore,

$$H = -2(0.5 \times \log_2(0.5)) = 1 \quad (1.20)$$

and 1 bit of entropy can be safely associated with the scenario of tossing a fair coin. What happens if instead a biased coin is used? Lets say heads is biased to have a probability of 0.6. Then $H = -(0.4 \times \log_2(0.4) + 0.6 \times \log_2(0.6)) \approx 0.97$. In the case of a biased coin, the entropy decreases as the amount of uncertainty in the system also decreased. Of course if the coin always lands on heads, there is 0 entropy in that system as it is certain what the result will be. With this being said Shannon's entropy amounts to determining the average uncertainty in a distribution.

A more conservative estimation of uncertainty is the minimum entropy of a probability distribution[38][25]. The min entropy is defined as

$$H_{min}(P) = -\log_2\left(\frac{1}{P_{max}}\right) \quad (1.21)$$

and is used to determine the degree of uncertainty in a system based off of the most probable value. As an example, think of guessing an individual's password. The min entropy would be associated with the probability of guessing the password correctly on the first try. Min entropy is especially relevant to the idea of extractors.

Extractors come up when attempting to differentiate between 'good' random bits and the output of a random number generator[38]. Good random bits are associated with being drawn from the uniform distribution while the output of a RNG may be some distance away from uniform. To bridge the gap between the two types of outputs, extractors are used. An extractor, E , is a function such that

$$E : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m \quad (1.22)$$

provided the distribution, X , associated with the bit string $\{0, 1\}^n$ has $H_{min} \geq k$. $\{0, 1\}^d$ is a uniform seed to the function. To summarize, if a distribution on X has min entropy k , the result is an output bit string $\{0, 1\}^m$ that is guaranteed to be ϵ close to uniform. An extractor is then parameterized by (n, d, ϵ, k, m) . The actual functions used to build extractors are typically derived from universal hash families[13], H , which contain a set of functions that map from A to B . Of particular importance are two-universal hash families that have the criteria

$$P_{h \in H}(h(x) = h(y)) \leq 1/|B|, \quad (1.23)$$

for all $x \neq y \in A$. Essentially, the probability of collisions for two-universal hash families is bounded by the size of the map destination.

While the utilization of an extractor derived from a two universal hash function has become common in privacy amplification schemes for QKD[38], the seed used for the extractor bears additional scrutiny. Similar to the use of a one time pad, the privacy of this seed is not necessarily preserved after use. This means that the extractor used cannot be readily re-used for multiple times without being compromised. To overcome this, the use of a strong extractor is needed. A strong extractor, E_s , performs the same functionality as an extractor except that the resultant distribution is also guaranteed to be ϵ close to uniform on the distribution that results from composing the extractors output with the original seed. In other words, $E_s(X, \{0, 1\}^d) \circ (\{0, 1\}^d) = \{0, 1\}^{m+d}$ is ϵ close to uniform. The important result of a strong extractor is that the information contained in the seed is preserved, meaning the extractor can be safely reused with the same seed. This is especially important for random number generation, as continuously updating a seed can undermine the output rate of the generator as it consumes random bits. This is especially true for a common extractor, the Toeplitz hash⁴, which requires $n + m - 1$ random seed bits to output $m < n$ uniform bits.

Fortunately, the extractors formed by two-universal hash families can be proven to be strong extractors with the help of the leftover hash lemma. The leftover hash lemma⁵ addresses the following:

$H = h_1, h_2, h_3, \dots, h_{2^d}$ is a two-universal hash family that maps from $\{0, 1\}^n$ to $\{0, 1\}^m$

X is a probability distribution on $\{0, 1\}^n$ with $H_{min} \geq k$

Then, for every $h_y \in H$ where $y \in \{0, 1\}^d$ acting on $x \in X$, the resultant distribution $h_y(x) \circ y$ is $\epsilon = 2^{-(m-k)/2}$ close to uniform. In other words, under these conditions all h_y 's acting on such an x form a $(n, d, 2^{(m-k)/2}, k, m)$ strong extractor. A formal proof for this lemma will not be given, instead the proof idea will be sketched out. Start with the assumption of a two-universal hash family with size 2^d . Determine the collision probability⁶ of the output of the extractor. Then, determine a bound on the distance, specifically the euclidean distance or L_2 norm, between the output and a uniform output. Finally, convert that notion of distance to a statistical measure of distance, where the statistical difference

⁴The Toeplitz hash comes from the GF_2 field, making it a two-universal hash family

⁵There are many forms to describe extractors, strong extractors, and the leftover hash lemma. I have been loosely following the definitions described in [38], while some readers may find alternative sources such as [57] easier to follow

⁶Collision probability refers to how likely the result of a hash on two separate inputs will map to the same output

between two distributions X and Y is defined as

$$\|X - Y\|^2 = \frac{1}{2} \sum_{v \in V} P(X = v) - P(Y = v) \leq \epsilon \quad (1.24)$$

1.5.2 Quantum Key Distribution and Weak Coherent Pulse sources

Every QKD protocol needs a source to produce the random states necessary in order to fulfill the goal of establishing a secure key. A QKD protocol can be performed with a variety of sources, including an entangled or quantum dot source. Specifically, for the QEYSSat mission a Weak Coherent Pulse Source (WCPS) is being developed in addition to an Entangled photon source (EPS). The focus of this thesis is on the WCPS, in particular, a WCPS similar to the one presented in [56], with one key difference (amongst others). In the airborne demonstration a repeating sequence of random states was sent meaning a limited amount of resources was devoted towards storage and transmission of the necessary data related to producing that sequence. For the QEYSSat Mission, a non-repeating sequence of random states will be used throughout the duration of a satellite pass. This switch has important ramifications - chief among them being the sheer volume of data being transmitted - that necessitates the development of a streaming software capable of mediating this transfer at the required rate. Additionally, an entropy source capable of producing enough random bits to turn into random states for the WCPS is required. This will be done with a QRNG and is the subject of [chapter 2](#). The development of the streaming software is the focus of [chapter 3](#). While the WCPS currently being developed for QEYSSat influenced the design of the software, many aspects to the software are independent of this potential WCPS. A brief overview of weak coherent pulses is given next.

Weak Coherent Pulses

As mentioned earlier in the introduction, weak coherent pulses can emulate a single photon source and have been successfully used in order to perform the BB84+ QKD protocol[27]. These pulses have their intensity and phase modulated in order to produce the desired state for the protocol. Due to the difficulty in creating single photon sources that produce random states at high enough rates, WCPS's have found themselves as a necessary but ultimately temporary intermediary as efforts continue to improve single photon sources. As the simpler counterpart, research into WCPS's has progressed to the point that there are now examples of WCPS's that are integrated on chip and operating in the gigahertz range[61].

Apart from this, the WCPS utilized in the experiment that would ultimately lead to the QEYSSat Mission, is shown below. This WCPS is effectively identical to what I will describe in [chapter 3](#) and is shown here to draw attention to where this project originated from.

QKD with WCPS

Recall earlier it was shown that the distribution of photons in a coherent state follows the Poissonian distribution. This had the important ramification that, in a given period t , 0, 1, or more than one photons can be observed. Even with the intensity brought down to the single photon level, there is still a non-negligible probability of multi-photon emissions in a particular finite weak coherent pulse. This opens WCPS's to photon number splitting attacks⁷ that target this aspect of a WCPS. To address this security risk, the decoy-state BB84⁸ protocol was developed. The overall protocol is the same as the original BB84, except now the random states sent have differing intensities correspond to signal states or decoy states. With this, any influence by Eve, including the previous PNS attack can be detected [27]

⁷PNS attacks are a special kind of intercept and resend attacks that target the multi-photon emissions to gain information about the random states that were sent

⁸Decoy state BB84 is sometimes written as BB84+

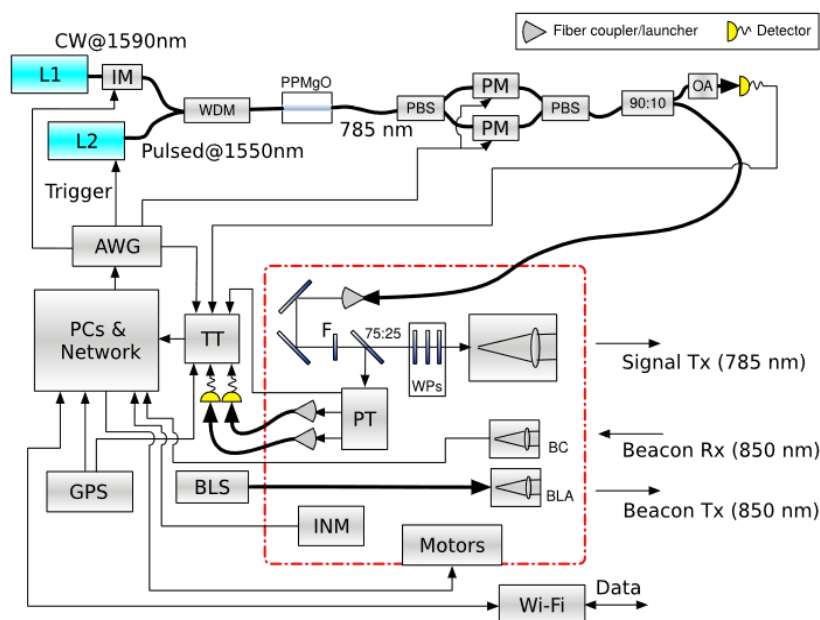


Figure 1.2: The WCPS system used in reference [56]. Figure taken directly from source. The main aspect to focus on for this project are the connections to and from the Arbitrary Waveform Generator(AWG). Three of the outgoing connections are what drive the WCPS. They go to the laser trigger, intensity modulator (IM), and phase modulator (PM) respectively. The last outgoing connection is to a time tagger (TT) to help identify the start of the repeating sequence and is more relevant to later phases of the QKD protocol. The incoming connection from the controlling PC is used to initially download the repeating sequence to the AWG's on board memory. This incoming connection is what changes the most when switching from a repeating to a non-repeating sequence. The remainder of the figure deals with intricacies of the WCPS like the transmitter apparatus that are independent of the streaming software to be discussed. For a more thorough discussion of this WCPS, refer to the original paper.

Chapter 2

Arrival Time Quantum Random Number Generator

In this chapter the QRNG scheme developed for the QEYSSat mission is presented. The chapter starts by going over the time of arrival type generators, in particular focusing on an implementation from [51] that served as a reference implementation to emulate and expand off of. Then the implementation, results, and next steps are described.

2.1 QEYSSat Requirement

Previous analysis done by the QPL team states, based off of the following assumptions, that the QRNG needs to be able to run at 220 Mb/s[21].

- 600 s of WCP data per pass¹
- 400 MHz source repetition rate [8]
- 4.4 random bits consumed per pulse (based off of signal/decoy ratio and algorithm responsible for generating biased bits from unbiased ones)
- 80 minutes minimum time between passes²

¹This is to accommodate the longest possible Satellite pass. Strictly speaking, all passes should finish in less than 10 minutes

²Corresponds to LEO orbit

4.4 random bits per pulse is a more subtle assumption that comes from randomizing the intensity and phase of the WCP states being produced for decoy state QKD. Currently, for QEYSSat the ratios for signal, decoy, and vacuum are 80%, 16%, 4% respectively[21]. All four polarization states are equiprobable. For randomizing phase, 2 unbiased bits are consumed. For randomizing intensity, there are 3 choices which could be represented with 2 bits if the probabilities were unbiased. Incorporating the bias results in generating a biased bit from unbiased ones. The current algorithm employed takes, on average, 2 unbiased bits to produce 1 biased bit. Additionally, the probability of selecting a decoy or vacuum state is conditioned on a signal state not being chosen. This amounts to an additional .4 bits (on average) consumed. In total

$$\begin{aligned}
b_{\text{phase}} + b_{\text{intensity}} &= 2 + (b_{\text{signal}} + b_{\text{non-signal}}) \\
&= 2 + (2 \times P(\text{Signal}) + (4 \times P(\text{Decoy}) + 4 \times P(\text{Vacuum}))) \\
&= 2 + 2(0.8) + 4(0.16) + 4(0.04) = 4.4
\end{aligned} \tag{2.1}$$

Where b_{phase} and $b_{\text{intensity}}$ corresponds to the unbiased random bits consumed per pulse for the randomization of the phase and intensity, on average. The software implementation of the above process responsible for taking a source of unbiased random bits and generating WCP states was done by Dr. Brendon L Higgins. I will motivate the implementation used for generating these WCP states. Each WCP state is stored in the low/high nibble³ of a byte. Recall that there are 4 phases (HVDA) and 3 intensities (signal, decoy, vacuum). Representing 4 options can be done with 2 bits, likewise for 3. So now, divide the nibble in half. The last 2 bits can be set to correspond to polarization states. The first 2 correspond to intensities. As these first two bits are only set when a signal state is not being created the following scheme is chosen: The numbers 0-3 correspond to signal states with random polarization. 4-7 correspond to decoy states with random polarization's. 8-11 corresponds to vacuum states with random polarization's. Now, selecting the intensity level requires generating biased bits from unbiased ones. Think of this as setting one of, or none of, the first two bits in the nibble. In other words initializing the WCP state to 0,4,or 8. An example algorithm is presented in Figure 2.1. The algorithm used in the QEYSSat mission is a more sophisticated version. Altogether, when a non-signal state is being created, 6 unbiased bits are used. When a signal-state is created, 4 unbiased bits are used. With this value of 4.4 bits per pulse, using the 400 MHz source repetition rate and 600 seconds maximum satellite pass duration yields a total of 1.056 Tb (132 GB) of random bits necessary for a pass. To generate this much random data between successive satellite passes leads to the previously stated value of 220 Mb/s.

³A nibble is four bits, or half a byte

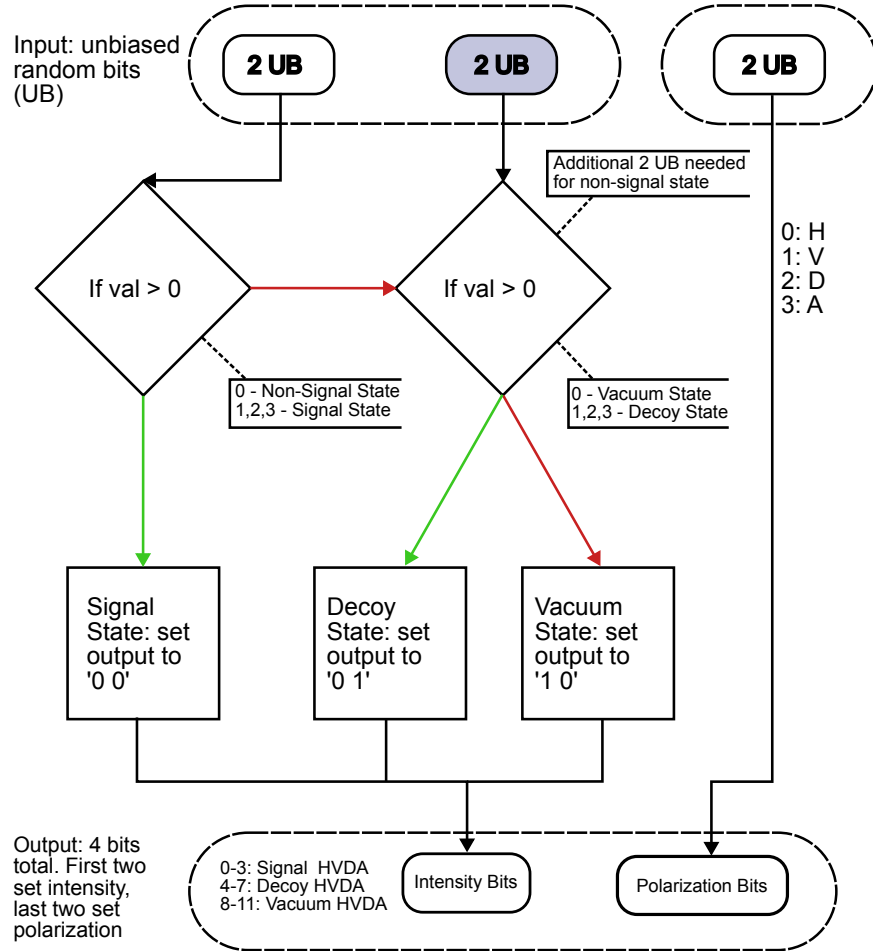


Figure 2.1: Two unbiased bits makes the values 0-3 equiprobable. For the specific case of 75% Signal 6.25% vacuum and 18.75% decoy one can envision a particular biasing algorithm to go from random bits to WCP states. Here, one would first read in 2 unbiased bits to randomize the polarization, and they would comprise the last 2 bits of the nibble. The first two bits are initialized to zero. Now, read another 2 bits and select between signal and non-signal states. If a signal state is chosen, do not set either of the first two bits of the nibble. If a non-signal state is chosen, read another 2 unbiased bits and determine which bit of the nibble to set. Altogether, this algorithm would consume 4.5 bits on average to produce a random WCP state.

2.2 "Time of Arrival" QRNG

Time of arrival (TOA) generators are based off of attenuated laser light being described by a coherent state. Since the photon number distribution of a random photon detection sequence is a Poissonian process, if one was to measure the arrival times of these photons with a detector the resulting distribution of detection pulses would also be Poissonian. By extension, if these arrival times were accumulated and the time difference between successive pairs determined, the resulting distribution follows the closely related exponential distribution. To demonstrate this, recall that for a Poissonian process with rate λ , the number of events that occur in a time t is proportional to λt . Then, the Poissonian process associated with λt average occurrences is

$$P(x) = \frac{(\lambda t)^x e^{-\lambda t}}{x!} \quad (2.2)$$

which implies the probability of 0 events occurring in a time t is, $P(x = 0) = e^{-\lambda t}$. Another way of interpreting this result is that the probability of the first occurrence at time T is greater than time t , $P(T > t) = P(x = 0) = e^{-\lambda t}$ [15]. Additionally, the probability that there is an event within time t is given by $P(T \leq t) = 1 - P(T > t) = 1 - e^{-\lambda t}$. The last result is the CDF of the exponential distribution. The previous discussion was centered around the arrival time and waiting-time for the events described by a Poissonian process with rate λ . By differentiating the previous result, one recovers the probability density function of the exponential distribution. Therefore, it is the exponential distribution that describes the inter-arrival time or waiting time distribution of the photons. The waiting time distribution is memoryless, which extends intuitively to being suited for random bit generation. This is because the Poisson distribution arises from independent and random events being emitted at some rate over a period of time. Since the system responsible for emitting these photons is described quantum mechanically, the resulting physical random number generator earns itself the classification of quantum random number generator.

As for the extraction of the random bits, this is done by examining the time deltas. To motivate this, recall that the earliest QRNG's based off of photons utilized a 50/50 beam splitter where each branch would lead to a detector [30][25]. If the photon hits detector A then a 0 would be output, if the photon hits detector B a 1 would be output. As there is no guiding influence on which path will be taken by the individual photons, assuming a perfect beam splitter, the result is a balanced number of 0's and 1's produced randomly. This can be expanded on by adding more consecutive beam splitters along each path. If for example 3 beam splitters are used, then there are four paths the individual photon can take and the output would be serialized as 00, 01, 10, or 11. This setup can be viewed as having

2 of the original QRNG's working together. However, both must be identical to each other and at most each QRNG is generating one bit per event. The issue of uniformity across the device is a big problem for this style of generator as detector efficiency mismatches as well as beam splitter imperfections can introduce bias into the output[25]. This same setup can be reproduced by removing the beam splitter and instead recording and examining time deltas for the events. If a delta falls before some specified time t , the result is a 0. Otherwise, assign a one. Deltas that exactly equal t should be discarded to avoid introducing a bias. This of course generates 0.5 bits per event, but can be expanded upon. Recall that the deltas follow an exponential distribution. There is a definite probability to how long one would expect to wait for the next photon to arrive. This can be taken advantage of to create multiple time bins that each delta could be placed in, analogous to the beam splitter case. If 3 times are specified, then 4 bins are created $(0, t_1), (t_1, t_2), (t_2, t_3), (t_3, \infty)$ and the output bits are derived from the binary representation of the bin that the delta is placed into, dependent on how many bits are necessary to represent all the bins. In other words, to extract n bits, one needs 2^n bins. Alternatively, $\log_2(k)$ bits are extracted from k bins. If the deltas could be recorded with infinite precision, then one could generate infinite bits from each event. Unfortunately, infinite precision is unattainable and the characteristics of the time tagging device as well as the detector limit the amount of bits that can be extracted from a delta. Specifically, the bin widths cannot be smaller than the detector jitter and time tagging resolution summed in quadrature.

$$b_{min} > \text{total jitter} = \sqrt{(\text{detector jitter})^2 + (\text{tagger jitter})^2 + (\text{tagger resolution})^2} \quad (2.3)$$

Where b_{min} refers to the minimum bin width.

One implementation of TOA generators that served as a clear reference due to its similarities with the generator that was implemented is from [51]. Here, deltas are unused and instead each arrival time, t_r is compared to an external time reference T . In this implementation the reference time T is 40.96ns and the resolution of the detection + time tagging apparatus is given as 160ps. This lends itself to the generation of exactly 256 bins leading to 8 bits per detection event. The use of an external time reference is to mitigate the bias inherently present in the exponential distribution. In contrast to this, the implementation that I used for QEYSSat will attempt to do two things. The first is to multiplex the rate by using an array of detectors, where each detector is itself a TOA QRNG. The second is to instead attempt to compensate the raw bias of the exponential distribution and electronics by introducing a characteristic curve, which effectively amounts to approximating the quantile function of the distribution. Other implementations instead focus on the poissonian nature of the incoming arrival times, and use that to toggle a device[32], similar to the scheme presented in [30]. Another pair of similar implementations

using TOA generators can be found in references [70][39]. Both of these references are related to each other. The first, [39] uses uniform bin width of 5 ps to generate time bins for the inter-arrival distribution and notes the large bias towards earlier bins due to the exponential nature of the distribution. As a consequence, the raw data is whitened to produce uniform random bits. The next, [70], addresses the exponential shape of the inter-arrival distribution by utilizing pulse shaping techniques to modify the incoming light source such that the incoming distribution is closer to uniform. This allows the majority of the uniform time bins from before to be directly used, no whitening is necessary. In both of these references, as well as [51], uniform time bins are used. Instead, the approach I examined looks to use non-uniform bin widths and work within the exponential shape of the incoming distribution.

Mathematical description

When describing the number of photons in a coherent state that are expected to be observed in some interval t the result is the Poissonian probability mass function [58][25]

$$P_k(t) = \frac{(\lambda t)^k e^{-\lambda t}}{k!} \quad (2.4)$$

Where λ is the rate or laser intensity, which when detectors are introduced, includes the detector efficiency η to become an effective rate $\lambda_{eff} = \eta \times \lambda$. k is the number of events expected, or in other terms the number of photons. Now, recall that the waiting time for a Poissonian process with a rate λ follows an exponential distribution with rate λ . This means the CDF, $F(t, \lambda)$ becomes $1 - e^{-\lambda t}$ and the quantile function for a particular probability is then specified to be

$$F^{-1}(p, \lambda) = \frac{-\ln(1 - p)}{\lambda}, 0 \leq p < 1. \quad (2.5)$$

This quantile function is what the characteristic curve attempts to imitate. As the exponential distribution is memoryless[18], the different intervals between quantiles is theoretically unproblematic as each interval is independent of the last. Now, the quantile function returns the time value associated with a particular probability. Currently, 64 bins are created from 65 bin edges (0 is the first 'edge'). n bins would then look like $(0, t_1), (t_1, t_2), \dots, (t_{n-1}, t_n)$ where each $t_1, t_2, \dots, t_{n-1}$ would theoretically correspond to $F^{-1}(\frac{i}{n}, \lambda)$, with i ranging from 0 to n . However, as the total population of samples (time deltas) is unknown during operation of the QRNG, the last bin extends to ∞ and the char

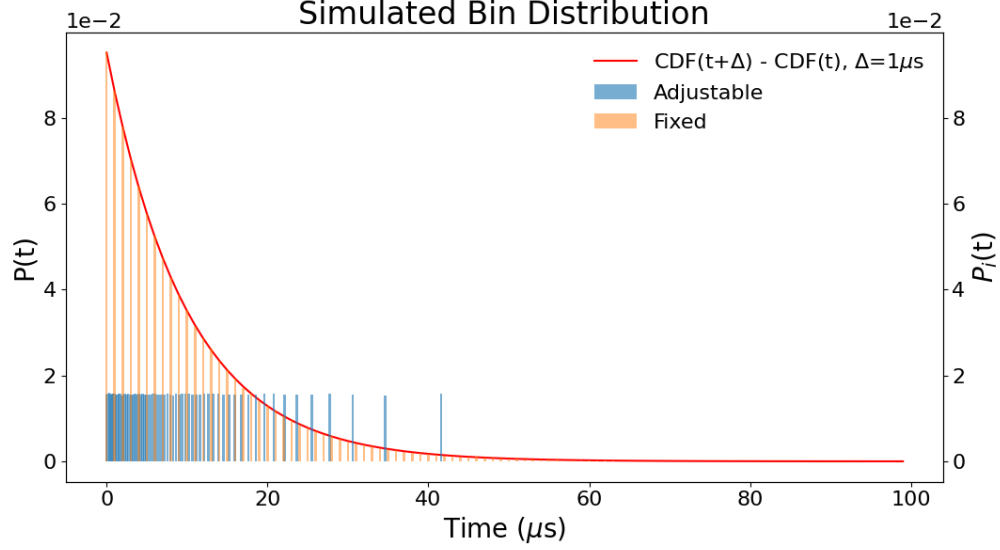


Figure 2.2: Simulated exponential distribution for events occurring at 100kHz rate. The 'fixed' bins correspond to 64 bins whose width were set to $1\mu\text{s}$, e.g the 'char curve' would correspond to $[0, 1, 2, 3, \dots, 63]$. To generate the quantiles, the ppf method associated with science pythons exponential distribution was used [69]. In a similar fashion, 1 million 'deltas' were generated using the random variates method of the exponential distribution object [69]. These deltas were then binned using both the fixed width bins and the adjustable bins derived from the quantiles. The left y-axis corresponds to a 'continuous' probability curve, generating by plotting the difference of the cumulative distribution function of the exponential distribution at two points, specified as Δ in the figure. The right axis is the individual probabilities of the i^{th} bin.

curve only determines values for $i = 0$ to $i = n - 1$. The imitation aspect of the characterization curve arises from the fact that the number of samples used to generate the bin values that comprise the curve is finite. A comparison between fixed width time bins and adjustable time bins derived from the quantile function is given in [Figure 2.2](#).

2.3 Implementation

2.3.1 Post Processing algorithm

Since this QRNG is meant to be comprised of several smaller QRNG's whose rate is multiplexed across the entire system, every step of the post processing is taken for each detector individually. This facet of the post processing will be implied and not explicitly stated. First, the raw time deltas are accumulated in an initial array. After the array is full, they are sorted and linear interpolation is subsequently applied on the sorted array to generate equiprobable quantiles that form the characterization curve. The characterization curve is an array holding bin edges. Each bin edge corresponds to its respective quantile. The first entry in the curve would cover deltas corresponding to a probability of 0 to $\frac{1}{k}$ where k is the number of bins. From there, bit generation begins. Every subsequent delta is placed into a bin by employing a binary search algorithm to find the index i of the first bin edge greater than the individual delta. The bin for that delta would then be the index minus one. Ergo, the zeroth bin corresponds to the first bin edge value. In the case of 64 bins, this equates to 6 bits per delta, and 000000 would be the output. [Figure 2.3](#) summarizes the process.

Regarding the linear interpolation, the algorithm used is sketched out next starting with a sorted array of input deltas belonging to a particular pixel QRNG. This sorted array will be referred to as rawDeltas.

Algorithm 1 Characterization Curve Creation Algorithm

procedure GENERATE CHAR CURVE

 x ▷ Used to generate index values in rawDeltas

 low, high

 linweight

for $i \leftarrow 1, < n$ **do**

$x = \frac{i}{\text{numBins}} \times \text{rawDeltas size}$ ▷ 'jump' forward in the array by a portion

 low = $\text{floor}(x)$

 high = $\text{ceil}(x)$

 linweight = $1 - \frac{i}{\text{numBins}}$

 charCurve[i] = (rawDeltas[high] - rawDeltas[low]) × linweight + rawDeltas[low]

end for

end procedure

The algorithm can be thought of as having placed all the deltas on an number line and jumping to the value corresponding to the i^{th} quantile. From here, working with the fact that `rawDeltas` only contains a sample of the total deltas that the QRNG will process, interpolate between the two values that would be a candidate for the i^{th} quantile. Repeat for all desired quantiles. After interpolation is complete for each quantile, the pixel now possesses a range of time values that can be used as time bins. Explicitly they would be: (`charCurve[0]`, `charCurve[1]`), (`charCurve[1]`, `charCurve[2]`), ... , (`charCurve[n-2]`, `charCurve[n-1]`) and the last bin, (`charCurve[n-1]`, `charCurve[∞]`) is implied if the binary search algorithm fails. This is because every delta must be greater than zero, and only deltas belonging to the last bin are greater than every entry in the characterization curve.

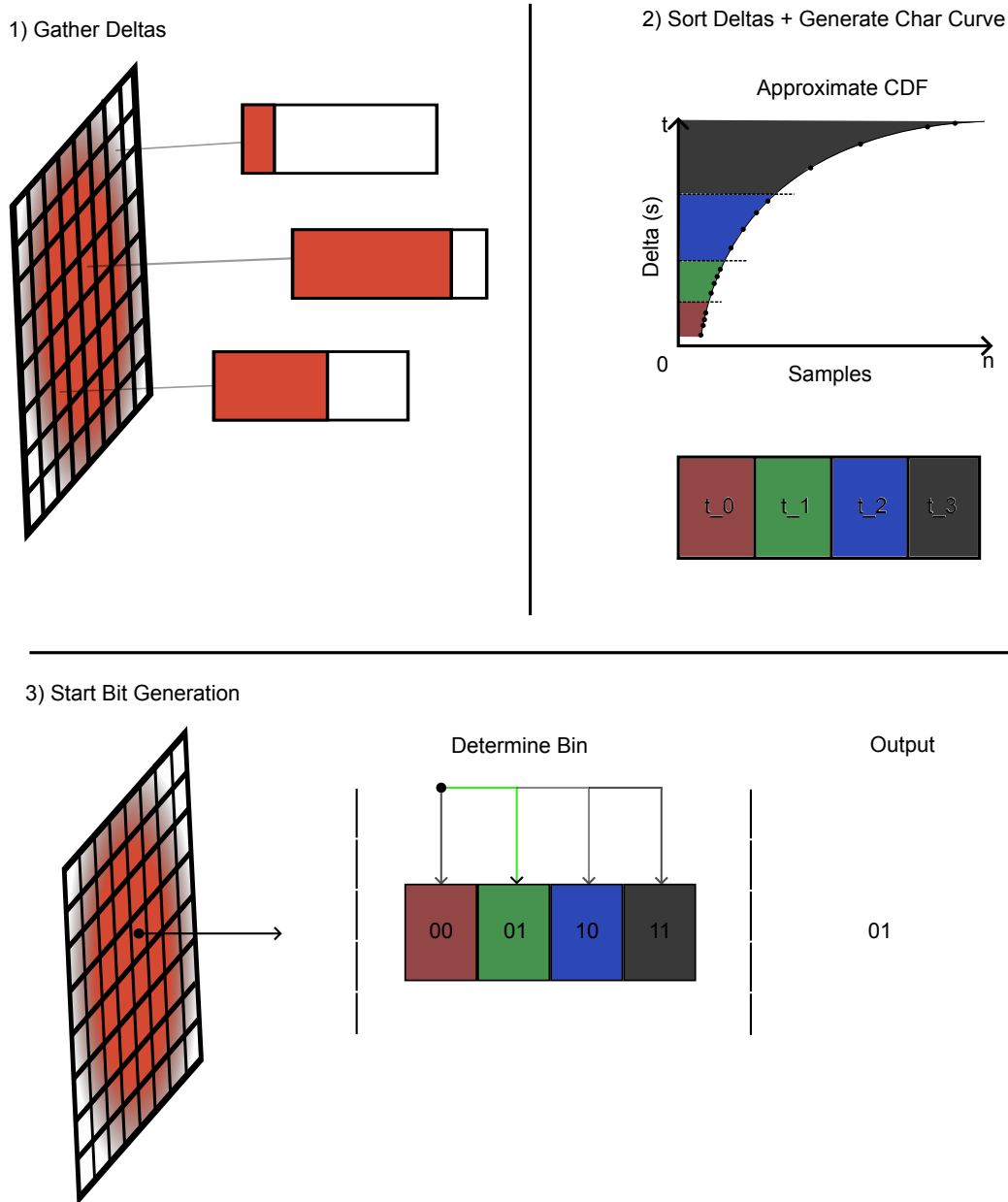


Figure 2.3: The TOA processing scheme for the QRNG is broken up into 3 distinct steps. First, a predetermined number of deltas for each SPAD are collected. Once collected, the characterization curve for that SPAD is calculated. Afterwards, any subsequent deltas are binned via the characterization curve and bits are output according to the index of the bin they are placed in. In the figure the photon was placed in the 2nd bin, with four total bins. Therefore, the output from the QRNG for that delta would be the binary value for the bin, encoded in $\log_2(k) = 2$ bits.

```

Here are the results of the single tests:

- The "Frequency" test FAILED both the analyses.

- The "Block Frequency" test FAILED both the analyses.

- The "Cumulative Sums" (forward) test FAILED both the analyses.
  The "Cumulative Sums" (backward) test FAILED both the analyses.

- The "Runs" test FAILED both the analyses.

- The "Longest Run of Ones" test FAILED both the analyses.

- The "Binary Matrix Rank" test FAILED both the analyses.

- The "Discrete Fourier Transform" test FAILED both the analyses.

- 2/148 of the "Non-overlapping Template Matching" tests passed both the analyses.
  146/148 of the "Non-overlapping Template Matching" tests FAILED both the analyses.

- The "Overlapping Template Matching" test FAILED both the analyses.

- The "Maurer's Universal Statistical" test FAILED both the analyses.

- The "Approximate Entropy" test FAILED the uniformity analysis.

- 8/8 of the "Random Excursions" tests FAILED both the analyses.

- 18/18 of the "Random Excursions Variant" tests FAILED both the analyses.

- The "Serial" (first) test FAILED both the analyses.
  The "Serial" (second) test FAILED both the analyses.

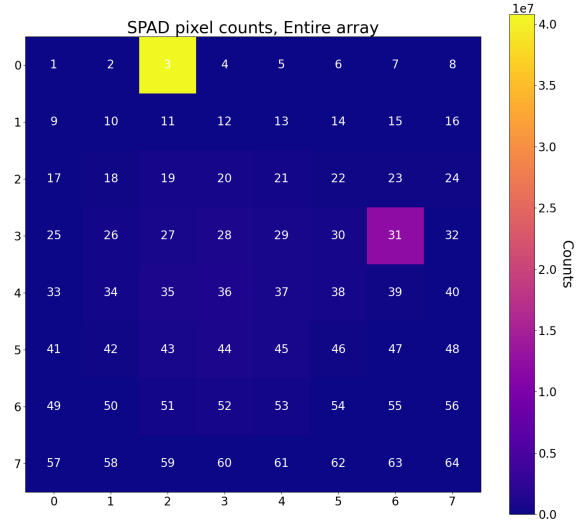
- The "Linear Complexity" test passed both the analyses.

```

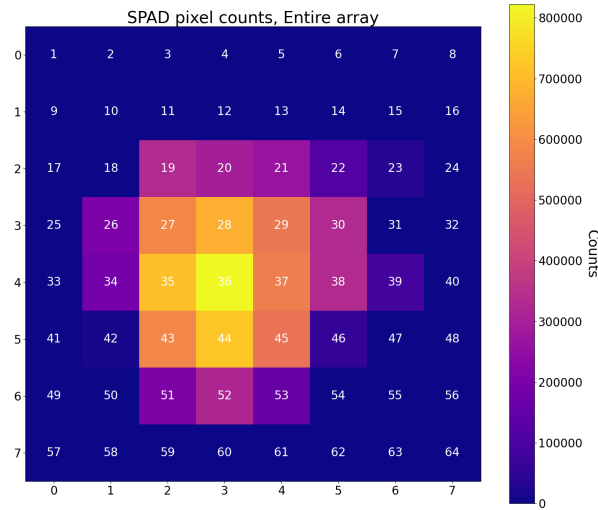
Figure 2.4: Initial Results of NIST STS show the output does not contain enough entropy to be considered random. Data used for char curve: 100kS, where S refers to samples or in this case time deltas.

2.3.2 Initial Results

To first test the algorithm data was collected from a device using an 8x8 SPAD array. For information on the detectors, refer to the appendix. The time stamps and addresses for photon counts from this device were then used to generate random bits using 64 bins. The resultant bits were then tested using NIST STS, a testing suite used to verify random bits are 'good'. [5] Since the standard for the QRNG, or any RNG for that matter, is to completely pass NIST STS the initial results in [Figure 2.4](#) show conclusively that there is some degree of predictability to the output of the QRNG. As the output is directly a result of the bin edges, they are investigated next. To do so three SPADs were selected. Two of which have a significantly higher DCR in comparison to the rest of the grid. These two hot pixels were selected to further examine the bin distribution as they had the most "counts" though many of them would be due to the dark counts. This would also serve as a rough approximation to the impact of dark counts on the resulting distribution.



(a)



(b)

Figure 2.5: (a) SPAD array heatmap of total counts. The two highest pixels have a significantly higher DCR contributing to them dominating the total counts. (b) Same as (a), but with the counts corresponding to pixel 3 and 31 set to zero.

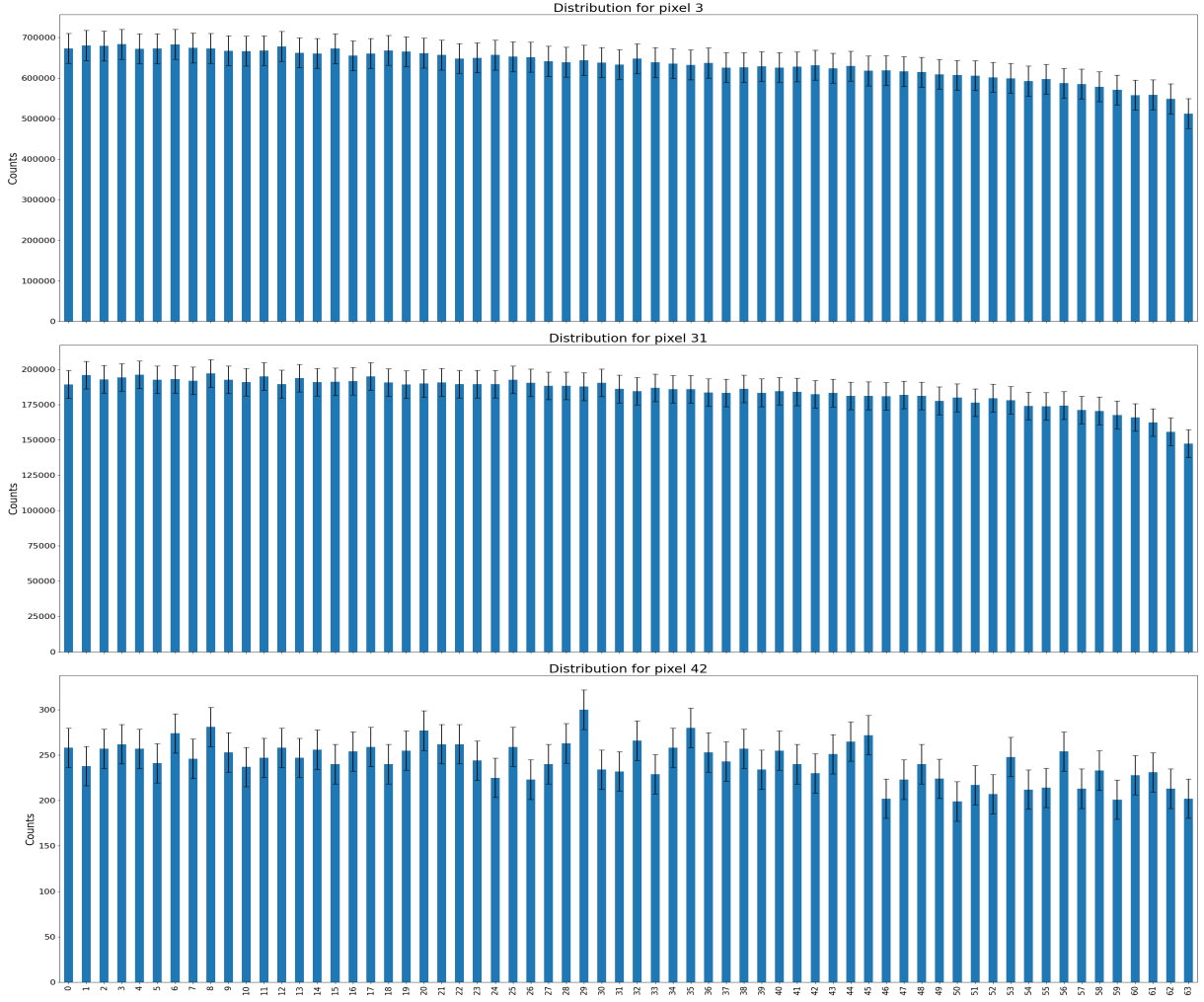


Figure 2.6: Bin distribution for select pixels of the test SPAD array. These bin counts are determined after the characterization curve is determined. Data was collected for 935 seconds. Total counts (including those used for the characterization curve) for pixels 3, 31, and 42 are 41585606, 12797432, and 115575 respectively. Number of deltas used to generate the characteristic curve: 1 million.

One key characteristic of the above bin distribution is that both pixel 3 and pixel 31 show a clear downwards trend in counts for later bins. A potential explanation would be the linear interpolation failing and/or being overly influenced by the amount of data used to generate the characteristic curve. To address this avenue, the amount of data taken in to generate the characteristic curve was varied from 100kS to 1MS to see if the characterization curve values vary or converge as the amount of input data is increased.

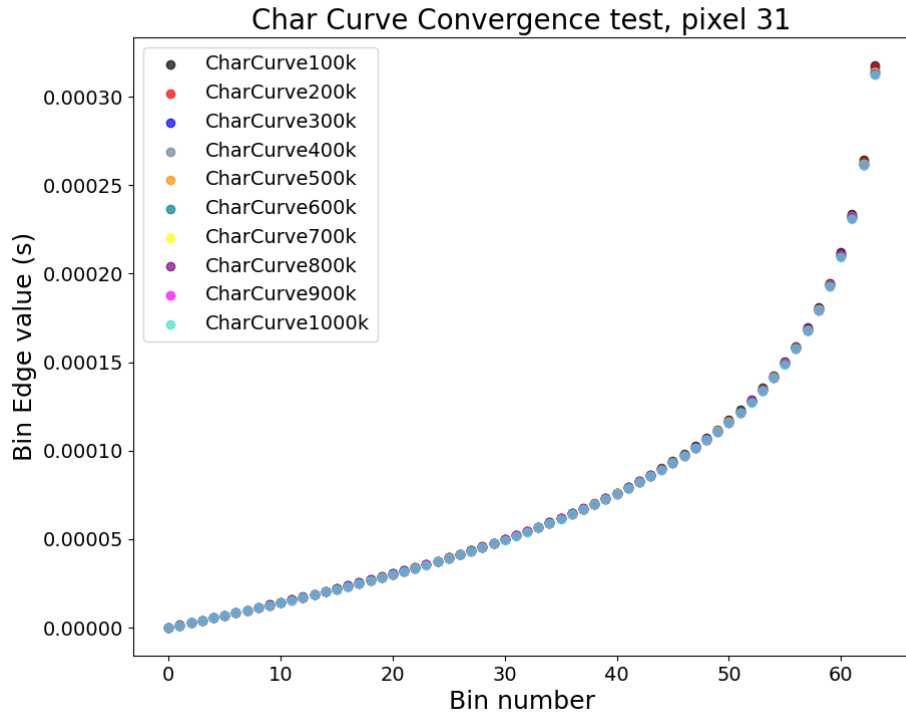


Figure 2.7: Results of varying the amount of data used per pixel to generate the characterization curve. For the later bins, especially the most extreme bins, as the amount of input data is increased the bin edge value trends downwards which translates to adjusting the sizes of the later bins. However, the earlier bins were mostly unaffected. Additionally, the resultant output QRNG bits continued to fail NIST STS.

Overall, varying the amount of data used to generate the characterization curve had a negligible effect except for the latest bins. Additionally, increasing the initial amount of data also increases the latency of the QRNG as the time for all individual pixels to start producing random bits also increases. So, since the amount of data used to generate the characterization curve had a net neutral effect it is likely not the variable responsible for introducing the bias in the generator. Next up was the interpolation itself. While linear interpolation is one avenue for generating these quantiles, this was done for a test data set ultimately originating from a file. Realistically, it would be preferable to accumulate these statistics on-line and adjust them as needed. This is the subject of streaming algorithms which is an ongoing research area. One recent development was the introduction of a t -digest data structure that has the ability to give highly-accurate estimates of quantiles, especially at the extremes (top 99th percentile for example)[16]. Increased performance over classic linear interpolation at more extreme quantiles was demonstrated. While the increased performance of quantile estimation is not necessarily a problem when only using 64 bins, as the number of bins increases to 1024 bins (10 bits per delta) the first bin edge would correspond to the top .0001 percent of the data. At these finite slices the shortcomings of the current linear interpolation may become more problematic.

The t -digest was used instead of the linear interpolation and the bin distribution was created again. The impact of the t -digest for the 64 bins scenario was negligible. To fully test the impact of the t -digest a larger amount of bins are necessary.

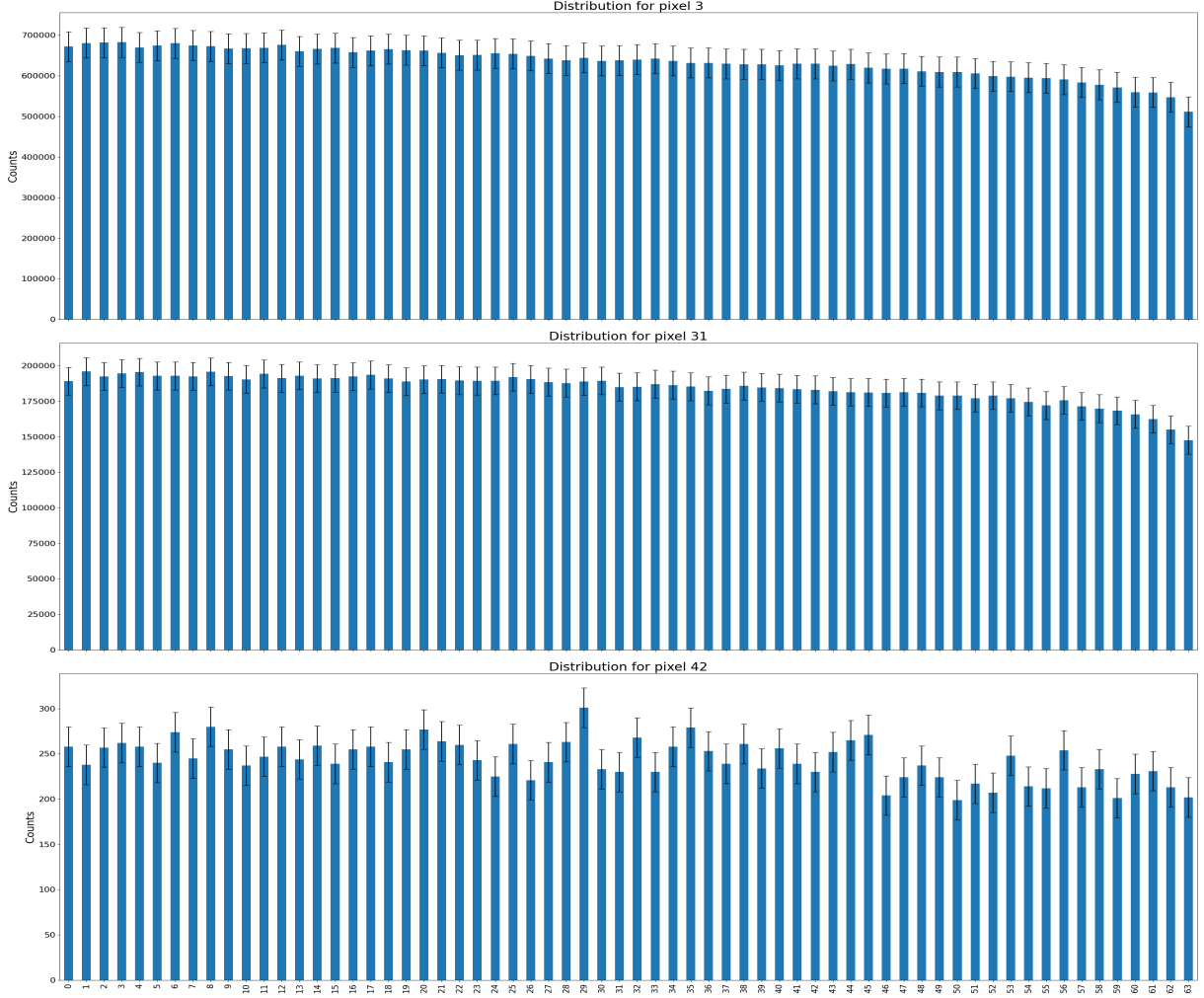


Figure 2.8: Recreating the same bin distribution as Figure 2.6 (same amount of input data to the t -digest as the previous interpolation). Overall the figure is practically identical to the previous and a similar bias towards earlier bins is observed. Data was collected for 935 seconds. Total counts (including those used for the characterization curve) for pixels 3, 31, and 42 are 41585606, 12797432, and 115575 respectively. Number of deltas used to generate the characteristic curve: 1 million.

The above investigation and subsequent adjustments centered on the characterization curve not the data being collected. Another potential avenue would be the impact of dark counts. The basis of this generator is that the incoming distribution of photons is Poissonian. However, with detector imperfections like dead time what is instead received is a distribution that is *nearly* poissonian. The accumulation of imperfections can be thought of as the system gaining some amount of memory about incoming events, and could undermine the underpinning of the post processing. However, the idea of approximating the distribution in real time is meant to address this at a high level. More relevant would be the impact of the dark counts. While in many cases the dark count rate is treated or modeled as poissonian, there has been dissension on if that is accurate[60][36][65]. If the DCR is poissonian, then the resultant mixing of photon counts with dark counts amounts to the convolution of two poissonian distributions which would also be poissonian and therefore not a problem for the QRNG. This is the case provided the dark count rate is suitably less than the incoming photon rate (which does not hold for the two hot pixels above). In the case where the DCR is not poissonian then the dark counts have a greater impact on introducing bias to the system as they are treated as photons but really come from a vastly different distribution. These results would indicate that the two hot pixels in the alternate SPAD system likely no longer follow the Poisson distribution for the dark counts. As these two pixels contribute the most to the outgoing bit stream, they are likely the driving factor behind the systemic failure of the raw bit stream to pass NIST STS. To accommodate the scenario of a weakly random/ non random output, the decision was made to implement a randomness extractor to take the weakly random raw data and turn them into a random bit stream. This is the subject of the next section.

2.4 Implementing Randomness Extractor

2.4.1 Min Entropy

Before being able to apply any kind of randomness extraction algorithm the amount of extractable random bits from the raw weakly random source must be determined and this is done through the use of min entropy. The min entropy of the raw bit string can be determined by the bin distribution. Recall that the min entropy is defined as

$$H_{min}(P) = -\log_2\left(\frac{1}{P_{max}}\right) \quad (2.6)$$

Which can be interpreted as looking for the most probable value of a distribution. This translates, for this particular QRNG, to the maximally probable bin. However, this also

means that one needs to process the random bits in chunks or else be forced into accumulating raw time tags for a specified duration which would certainly create memory problems for the underlying software. The chunk size is set to be a power of 2 for convenience, and larger powers of 2 are preferred to get closer to the uniform distribution that is desired. I follow the prescription set by Xiongfeng Ma [38] which deals with an input bit string of length m with min entropy k and security parameter ϵ (from the leftover hash lemma - see introduction subsection 1.5.1 on randomness extractors).

1. Set the output length to be $m = k - \log_2(\epsilon)$
2. Construct a Toeplitz matrix of size $n + m - 1$ from a random seed
3. Apply Toeplitz hash via multiplication of the Toeplitz matrix with the raw random bit string m .

2.4.2 Toeplitz Extractor

Any extractor, in the context of randomness extraction, takes a raw bit string and combined it with a bit seed to output an almost perfectly uniform distribution of bits. A strong extractor does the same thing, but with the additional caveat that little to no of its private seed is exposed[38]. This has the important implication that the seed can be re-used. In the case of a Toeplitz matrix this is very important as constructing an $n \times m$ Toeplitz matrix from $n + m - 1$ bits is a net loss in total entropy generation. As for the extraction itself, it amounts to a matrix-vector product between the Toeplitz and the raw bit string. This product, algorithmically, has a $O(n^2)$ time complexity which is quite severe.

As an aside, the Toeplitz hash can be proven to be a strong extractor with the use of the leftover hash lemma, see subsection 1.5.1.

Computational Speedup

The Toeplitz or diagonal-constant matrix is a structured matrix with the additional condition that all of its diagonals must have the same value[22]. Formally, for a Toeplitz matrix T whose entries are denoted $T_{i,j}$

$$T_{i,j} = T_{i+1,j+1} \tag{2.7}$$

Provided the element at $(i + 1, j + 1)$ exists. Below is an example of a Toeplitz matrix

$$\begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ b_0 & a_0 & a_1 & a_2 \\ b_1 & b_0 & a_0 & a_1 \\ b_2 & b_1 & b_0 & a_0 \end{pmatrix}$$

Where it can also be observed that a Toeplitz matrix is completely defined by its first row and column. Since both row and column share one entry, generating a Toeplitz matrix requires only $n + m - 1$ bits[38]. The Toeplitz matrix is not necessarily square, but it is similar to another type of structured matrix that is square. The circular matrix, C, can be thought of as a specialization of the Toeplitz matrix where not only are all diagonal entries the same but each row and column are right permutations of the first[22]. Formally,

$$C_{i,j} = C_{i+1,j+1} = C_{i-j} \quad (2.8)$$

$$C_i = C_{j-i} = C_{i-j} \quad (2.9)$$

Below is an example of a circular matrix

$$\begin{pmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 \\ a_5 & a_0 & a_1 & a_2 & a_3 & a_4 \\ a_4 & a_5 & a_0 & a_1 & a_2 & a_3 \\ a_3 & a_4 & a_5 & a_0 & a_1 & a_2 \\ a_2 & a_3 & a_4 & a_5 & a_0 & a_1 \\ a_1 & a_2 & a_3 & a_4 & a_5 & a_0 \end{pmatrix}$$

In this case the Circular matrix is completely described by either its first column or row. What is especially important about circular matrices is that they are diagonalized by the Fast Fourier Transform (FFT). This allows us to use much faster algorithms for computing the matrix vector product for the Toeplitz extractor that take $O(n \log(n))$ time complexity [53]. The first step to doing that is to embed the Toeplitz matrix T into a circular matrix C. Embedding a Toeplitz matrix into a circular matrix is very straightforward. Recall a Toeplitz matrix is defined by its first row and column. Take the column, transpose it, and append the first value to the end of the first row. Next, work backwards through the remainder of the transposed column and add these values to the first row. Now the $n \times m$ Toeplitz matrix is embedded in a $(n+m) \times (n+m)$ circular matrix. To see this lets sketch out the original problem. The goal is to construct a circular matrix C that has a Toeplitz

matrix T embedded in it. By the structure of the circular matrix the overall structure must look like the following:

$$\begin{pmatrix} [T] & [A] \\ [A] & [T] \end{pmatrix}$$

Where T is the Toeplitz matrix to be embedded and A is a currently undefined matrix to be determined whose entries will be denoted $x_{i,j}$. Lets use the 4x4 Toeplitz from earlier as our T and fill in the known entries of the circular matrix.

$$\begin{pmatrix} a_0 & a_1 & a_2 & a_3 & x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} \\ b_0 & a_0 & a_1 & a_2 & x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} \\ b_1 & b_0 & a_0 & a_1 & x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} \\ b_2 & b_1 & b_0 & a_0 & x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} \\ x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & a_0 & a_1 & a_2 & a_3 \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & b_0 & a_0 & a_1 & a_2 \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & b_1 & b_0 & a_0 & a_1 \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} & b_2 & b_1 & b_0 & a_0 \end{pmatrix}$$

Next, lets solve for as many entries of A as possible using the properties of a circular matrix. We can determine $x_{0,2} = x_{1,3} = b_1$, $x_{0,3} = b_0$, $x_{2,3} = x_{1,2} = x_{0,1} = b_2$, $x_{2,0} = x_{3,1} = a_2$, $x_{1,0} = x_{2,1} = x_{3,2} = a_3$. That leaves only the diagonal entries of A unsolved. Since these entries are independent of all other ones we are free to assign whichever value we want. In this case, we will go with a_0 . Now, the final Circular matrix is

$$\begin{pmatrix} a_0 & a_1 & a_2 & a_3 & a_0 & b_2 & b_1 & b_0 \\ b_0 & a_0 & a_1 & a_2 & a_3 & a_0 & b_2 & b_1 \\ b_1 & b_0 & a_0 & a_1 & a_2 & a_3 & a_0 & b_2 \\ b_2 & b_1 & b_0 & a_0 & a_1 & a_2 & a_3 & a_0 \\ a_0 & b_2 & b_1 & b_0 & a_0 & a_1 & a_2 & a_3 \\ a_3 & a_0 & b_2 & b_1 & b_0 & a_0 & a_1 & a_2 \\ a_2 & a_3 & a_0 & b_2 & b_1 & b_0 & a_0 & a_1 \\ a_1 & a_2 & a_3 & a_0 & b_2 & b_1 & b_0 & a_0 \end{pmatrix}$$

Whose defined by the row $[a_0, a_1, a_2, a_3, a_0, b_2, b_1, b_0]$. The original T was defined by the row $[a_0, a_1, a_2, a_3]$ and the column $[a_0, b_0, b_1, b_2]'$. From here it should be clear where the original prescription arises from. If I were to take the column, transpose it and add the first value to the original I would get $[a_0, a_1, a_2, a_3, a_0]$. Next, I would work backwards through the remaining values of the column and append them retrieving the original result for the row defining C. Of course, you can swap rows with columns in the prescription and achieve the column defining C in the same way. With this circular C the original matrix vector product to perform the extraction can be calculated in the following steps:

1. Embed T of size $m \times n$ in circular C
2. Compute DFT of C (this amounts to performing DFT on the row or column that defines C)
3. Compute DFT of raw bit string n
4. Perform matrix vector product
5. Compute inverse DFT on resulting vector
6. Extract first m random bits of result

It should be noted that a DFT (Direct Fourier Transform) is what is calculated by a FFT, with the caveat that a FFT refers to calculating a DFT with a fast algorithm. Since each FFT takes $O(n \log(n))$ time the overall process takes $O(n \log(n))$ time[53]. A sizable improvement over the typical $O(n^2)$.

2.5 Results

The entire post processing scheme depicted in Figure 2.9 was implemented in C++ on a laptop running an Ubuntu distribution, far from an optimized system. With this in mind the rate of the QRNG is not determined or stated in the subsequent analysis as any values given would not be representative of the actual generator. In addition, Toeplitz extraction is quite slow (A few MB/s) for a non-embedded environment[38]. As the extraction is the most computationally intensive part of the algorithm, any value given would essentially be just for the extraction.

To integrate the extractor in the QRNG the min entropy is fixed. This serves two purposes, the first is to allow the potential optimization of the FFT by using the more exhaustive planner options in the library like its patient or exhaustive mode[20]. This comes at the downside of introducing a non-negligible additional delay into the QRNG as these differing modes can take several seconds to complete. The second reason is related to the online operation of the QRNG. By online I mean after all the individual QRNG's are running and all characterization curves have been made. Non-constant FFT parameters demand reconstruction of all aspects of the Toeplitz extractor for every block of random bits being processed. This would entail near constant memory deallocation and allocation, which is computationally expensive. As for the process of fixing the min-entropy there are two avenues. The first would be to characterize the system and determine the lower bound

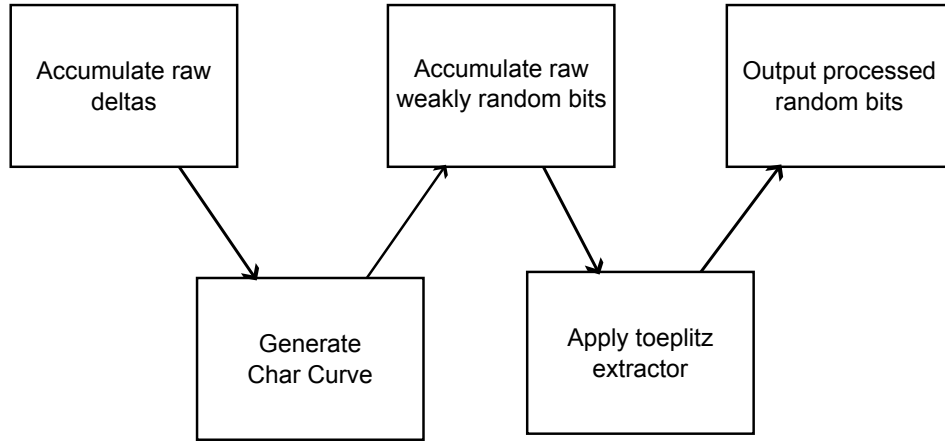


Figure 2.9: The entire post processing scheme for the QRNG.

of the min entropy for that specific system. As for right now, the process for integrating the above QRNG post processing into a more embedded environment like the Photon-to-digital converter (PDC) described in [12] is ongoing, this inhibits fully characterizing the system to determine the lower bound of the min entropy. Instead, a heuristic approach was chosen where the min entropy for several blocks of raw data was determined, and a value slightly lower than the minimum of those sample points was chosen. During operation, before processing the raw block the min entropy is calculated and compared to this fixed value. In the event that the entropy is lower a one time FFT is executed with specific parameters.

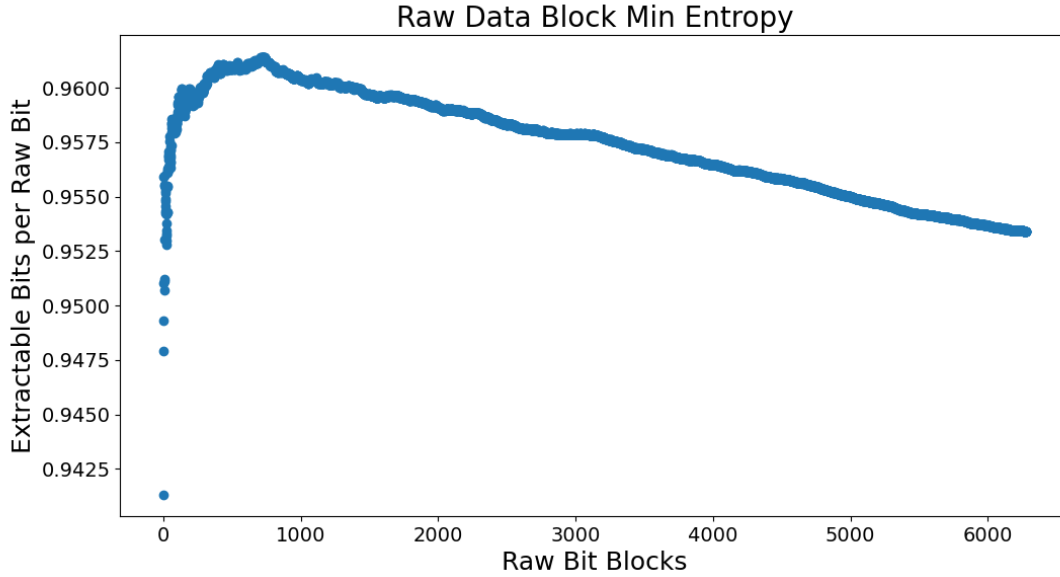


Figure 2.10: Min entropy of raw bit blocks. Each point on the graph represents the min entropy calculated from one block of 2^{16} bits, as a fraction of extractable bits. The calculation was performed by determining the maximally probable bin for that block of data, and then dividing that by the theoretical max value which is dependent on the number of bins. In the case of 64 bins, the theoretical maximum is 6. As an example, suppose the maximally probable bin had a probability of .01875 ($\frac{1.2}{64}$). The min entropy calculation would give a value of 5.73, which when divided by 6 gives ≈ 0.95 random bits per raw bit. So for that block of data, 62259 bits could be extracted. The processed data corresponds to the same dataset mentioned earlier, where raw time tags were collected from an 8x8 SPAD array not to be confused with the PDC implementation that is ongoing.

Based off of [Figure 2.10](#), the min entropy was fixed to be 0.91. In the event multiple one time FFT's are called this value can either be lowered or this could be taken as a sign to re-determine the values for the characterization curve. As the conditions for the physical system, like the dead time of the SPADs or the intensity of the photon source can fluctuate over time, this can render the previously determined characterization curve inaccurate. To compensate for this, new time tags can be accumulated and a new characterization curve rendered. This is another place where the *t*-digest shines, as it has the ability to merge digests together which avoids completely throwing out the original characterization data. This allows the system to adapt when run for longer periods of time. It should be noted that data used to make or change the characterization curve cannot be subsequently processed to avoid introducing a bias. One can already observe some potential degradation of the characterization curve values as there is a definite trend downward in the min entropy of the raw bit blocks as more blocks are processed with the same characterization curve.

With the inclusion of the extractor, the bin/byte distributions are re-examined to quantify the impact of the extractor. [Figure 2.11](#) captures the results. In both figures it can be observed that the overall distribution is closer to the theoretical red line with a tighter spread. As a token example, look at bin 60 or the zeroth byte. For both of those examples they were previously noticeably more probable prior to applying the Toeplitz extractor.

While there are still bytes that are noticeably more probable than their peers it is also important to take into account which bytes that is. The Toeplitz extractor that was implemented was meant to produce a uniform bit string - meaning an equal number of 0's and 1's. In [Figure 2.11](#) one of the highest peaks in the 'after' plot of the byte distribution corresponds to the number 15, which when represented with 8 bits is 00001111. From the perspective of generating a uniform sequence of 0's and 1's having this byte be slightly more probable is acceptable, as long as it does not appear periodically. Additionally, 2^{16} bits corresponds to only ≈ 10923 deltas, so there is some degree of deviation from uniform to be expected.

While the inclusion of the Toeplitz extractor has a visible impact on the bin and byte distribution, the next step is to verify that the produced bits are certifiably random by passing NIST STS. So, the outgoing bit stream was again tested with NIST STS and the results are shown in [Table 2.1](#).

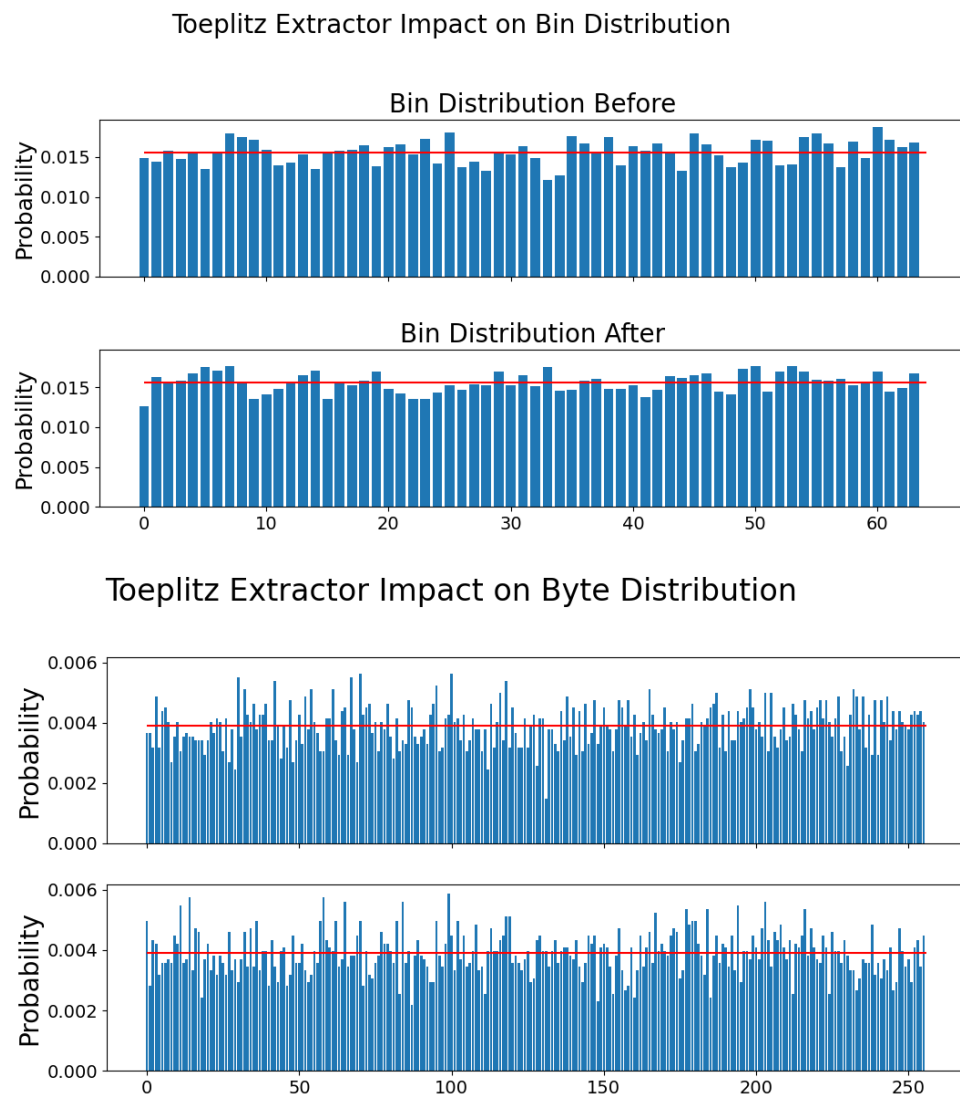


Figure 2.11: Bin and Byte distribution before and after applying Toeplitz extractor. All plots derived from the same block of data. The red line corresponds to the theoretical result where all bins or bytes are equiprobable.

NIST TEST ITEM	p value	Result
Frequency	0.554420	Pass
Block Frequency	0.032923	Pass
Runs	0.719747	Pass
Cumulative Sums	0.699313	Pass
Longest Run	0.213309	Pass
Rank	0.191687	Pass
DFT	0.851383	Pass
Non Overlapping Template	0.595549	Pass
Overlapping Template	0.334538	Pass
Universal	0.202268	Pass
Approximate Entropy	0.699313	Pass
Random Excursions	0.316916	Pass
Random Excursions Variant	0.386280	Pass
Serial	0.437274	Pass
Linear Complexity	0.080519	Pass

Table 2.1: NIST STS results after processing raw bits with Toeplitz extractor

The impact of including an extractor is now verified as the processed bits can be considered random unlike before in [Figure 2.4](#). It should be noted that all the effects of extractor were analyzed using the same data set that generated all the figures presented in this chapter.

2.5.1 Characterization Curve and Quantile Function

Using the total counts and the fact that data was collected for 935 seconds, one is able to determine an approximate rate for a particular detector. With an approximate rate in hand, the quantile function, [Equation 2.5](#), can be plotted against the characterization curve for each pixel. In the case of pixel 31, 12797432 total counts over 60 seconds yields an approximate rate of 13.7kHz. [Figure 2.12](#) shows that the characterization curve almost matches the expected shape of the quantile function, [Equation 2.5](#), for this approximate rate. Additionally, estimating the rate parameter of the exponential distribution through maximum likelihood estimation (MLE), which for the exponential distribution is the inverse of the mean for the sample data, also agrees with the characterization curve values. The rate determined by MLE, 13.8kHz, is also fairly close to that pixels dark count rate of 14.4kHz. One can also see the 9th to 25th bin edges are a little higher than theoretically expected, this could explain why many of the earlier bins are slightly more probable than their later counterparts. Addressing the difference between the characteristic curve and the quantile function should have the positive effect of raising the min entropy and potentially alleviating the need for an extractor.

2.6 Outlook

This QRNG is based off of the inter-arrival time of photons which follow the exponential distribution as the arrival time of photons follow the Poissonian distribution. This exponential distribution is naturally skewed to favor earlier times than later ones, which leads to an inherent bias when attempting to bin the raw deltas for bit generation. Additionally, real world imperfections such as detector dead time and afterpulses cause the measured data to be nearly Poissonian which can be attributed to the system gaining some fictitious memory about the presence of a photon, thereby expecting another photon to arrive soon. To account for this, a characterization curve was introduced to approximate the real time quantiles of the incoming distribution to attempt to create bins that were equiprobable. As shown in [Figure 2.6](#) and [Figure 2.4](#), the characterization curve was somewhat successful but ultimately did not produce uniformly random bits.

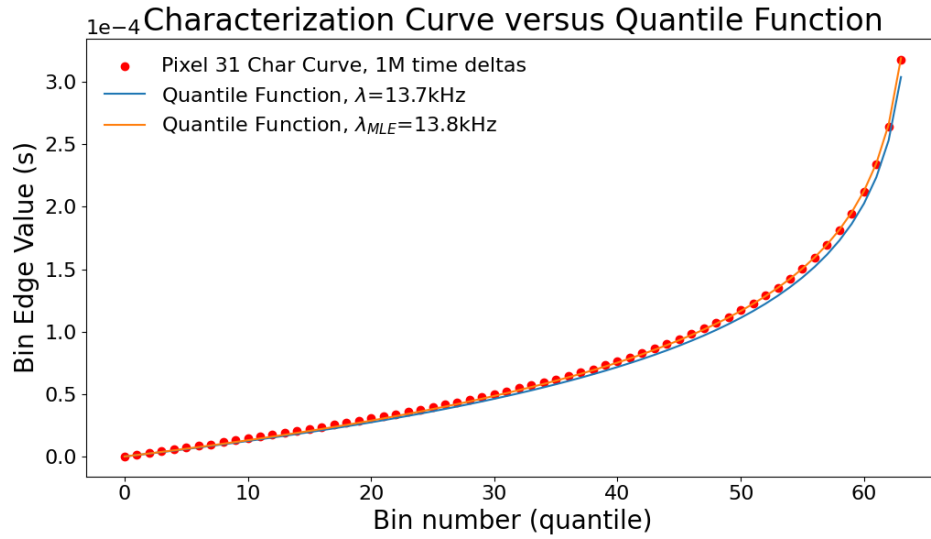


Figure 2.12: Characterization curve values for pixel 31 compared to quantile function [Equation 2.5](#). An approximate rate of 13.7kHz was determined and used to plot the quantile function across the range of percentiles (0 to 98) that the characterization curve would cover. The output of the quantile function would be the theoretical time value that particular bin edge should be near. The ideal situation would be to have the data points on the blue curve. Note that this would cause earlier bins to be smaller (less probable) and later bins larger (more probable).

To work towards producing certifiably random bits, the characterization curve was examined. First by investigating how the amount of data used to create the bins as in [Figure 2.7](#) and determining if the values converged as the amount of data was varied, or if they were relatively constant. It turned out that only the later bins were noticeably effected, with the last bin becoming larger as more data was used. However, in each case the resultant output bits continued to fail NIST STS. Additionally, the linear interpolation used in the creation of the characterization curve was scrutinized by using an alternative method of approximating quantiles, the t -digest[\[16\]](#). [Figure 2.8](#) showed a minimal impact of the t -digest in addressing the skewed probability towards earlier bins. [Figure 2.12](#) showed that the characteristic curve almost matches the quantile function of the approximate rate determined by taking the total counts for that pixel and dividing it by the collection time. The small deviation present in the figure does imply earlier bins being more probable than later ones, which was shown in [Figure 2.6](#). Closing the distance between the theoretical and measured characteristic curve should improve the binning of the QRNG. Currently, to address the failing of the characteristic curve to produce equiprobable bins, a Toeplitz extractor was implemented to further process the raw bits. The result was the production of bits that could pass NIST STS.

The use of a characterization curve is unique among time of arrival style QRNG's, which allows the device to compensate for detector imperfections as well as non-uniformity of detectors across an array of SPADs. Additionally, it is also possible to adjust the characterization curve on the fly when taking advantage of the t -digests ability to merge distinct digests. If the performance of the QRNG begins to degrade, a new digest could be constructed and merged with the original in order to generate new bins that take into account more recent deltas. As the entire QRNG is comprised of several smaller QRNG's, the correction of characterization curves could be done one at a time, instituting a minimal hit to the overall generation rate. The one downside to the use of a characterization curve is that there is a non-negligible ramp up period as the individual QRNG's start producing random bits.

Lastly, the original goal for the development of this generator was to make a QRNG that could run at 220 Mb/s. At the time of this writing only the highest performing chips commercially available from ID Quantique, specifically the Quantis QRNG PCIe model that advertises 240 Mb/s entropy generation, could meet this requirement. On the other hand, very expensive custom chips have been engineered and published that have rates well into the Gb/s regime[\[11\]\[32\]](#). As the implementation of the presented algorithm into a physical system is still ongoing⁴, the rate of this generator is unknown. However, the ability of this generator to meet the baseline rate required by QEYSSat is very feasible based off of

⁴The potential implementation would be using the PDC described in [\[12\]](#)

what is present in the literature, even when using hardware that wasn't designed with this particular algorithm in mind[12]. In the short term, the algorithm only requires three pieces of information. The first is the time stamp of the photons. Second, is the address of which detector the tags correspond to. Lastly, the resolution of the device is needed. Therefore, data from virtually any setup with those three components can have a finite number of certified random bits produced. From here a typical method of random expansion can be employed where the output random bits are used to seed a CSPRNG⁵. One open source library that should be suitable for this type of expansion would be OpenSSL[4]. As an initial test, data read from the output of the QRNG ($\approx 40\text{MB}$) was expanded to 64 GB of random data at a rate of $\approx 1.98\text{Gb/s}$. Only 7.8 MB of the 40 MB input file was used. Taking the performance of the QRNG on a personal laptop (about 1 Mb/s), with 2 Gb/s implies a total time of about 14 minutes to generate 1.056 Terabits of random data.

2.6.1 Improvements

Even with the feasibility of meeting the QEYSSat baseline requirement in mind, there are still a few areas for improvement. The first is the Toeplitz extractor itself. Figure 2.10 showcases a wide range of values from just above 0.92 to just shy of 0.98. Improving the initial processing via the creation of the characterization curve could raise the minimum value of the min entropy which in turn would increase the amount of random bits that are extracted. To improve the characterization curve, one could investigate two additional sources of bias for the system. The first is directly related to the characterization curve, afterpulses. SPAD afterpulsing is inherently a correlated process that could also be involved in earlier bins being more probable as shown in Figure 2.6. Being able to reliably identify and eliminate potential tags corresponding to afterpulses would improve the initial phase of the post processing. Related to this, is cross talk in the array. Cross talk refers to the occurrence of a click in SPAD B due to a click in its neighbor SPAD A. In other words, for an array of SPAD's there is a non zero probability of a count in one SPAD triggering its nearest or next-nearest neighbors. This could cause the characterization curves to be malformed for multiple SPAD's, or for deltas to be determined from the combination of two counts that were not due to an incoming photon. Mitigating the impact of cross talk on the QRNG would be nontrivial. One naive approach could be to determine the nearest neighbors for a SPAD when a count is received, then discard all counts for its nearest neighbors for a predetermined window. The main downside is the loss of counts,

⁵A CSPRNG stand for cryptographically secure PRNG as many PRNG's like the Mersenne twister can be proven to be insecure. These CSPRNG's have their output held to a higher standard to be used in applications where security matters.

however if the result of explicitly taking into account imperfections of the SPAD array is that the Toeplitz extractor is no longer necessary then the overall rate of the QRNG would likely increase. Otherwise, using the extractor is preferable over increased complexity of the underlying software with no real benefit to improving the throughput. Additionally, the current security parameter of the extractor is well above what is necessary. Larger block sizes of the FFT can be less efficient [20], therefore choosing small block sizes for the random bits may help improve the throughput of a potential embedded implementation of the generator.

Hybrid Algorithm

Another avenue to improve the performance would be the inclusion of an additional parallel processing. The QRNG as described above looks at the temporal distribution of SPADs and extracts bits from the inherent randomness present in that regime. However, the spatial locality of the photons is also indeterminate until a SPAD is triggered. This idea of utilizing the spatial distribution of photons has been used to create a QRNG, where a singular micro channel plate (MCP) was used as a single photon detector[74]. This detector had a square front facing plate of a specific size and the ability to determine the position of photon arrival by determining the centroid of the electron cloud that results from a photon detection. The square plate was subdivided into a grid of evenly sized squares. The output of the setup was dependant on which square the photon arrived in.

Instead of extracting the position from the centroid, one could use the address of the SPAD in an array as an equivalent method. This removes the constraint of calculating the centroid and instead the physical extent of the SPAD can be used. In this case the size of the array directly translates to how many bits can be extracted. The downside being that detector uniformity and intensity across the array's would become much more important. Additionally, while correlations between space and time can be introduced into optical systems, instead there would need to be a suppression of any incidental correlations[26]. This could be non-trivial. However, if done the same counts could be processed by the photon time of arrival and the photon spot of arrival in parallel. The output of each processing could be subsequently combined once processed.

For the case of an 8x8 SPAD array, like that in the physical system being used to implement the QRNG, 6 bits per detection event could be extracted and combined with the 6 bits per delta. Effectively doubling the rate of the QRNG assuming both processing methods could be computed in parallel at the same time.

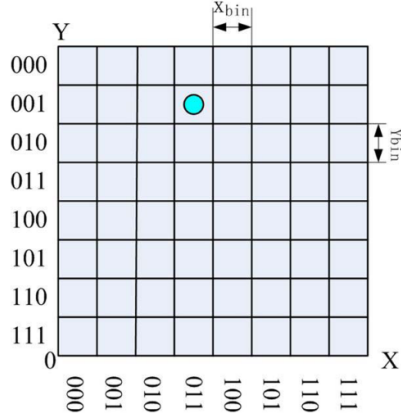


Figure 2.13: Binary code for spot-of-arrival QRNG. Figure taken directly from [74] In this case the output of the generator would be 001011. This generator's rate is limited by the spatial resolution of the setup, which forces the detector to be run at a slower rate. Giving it an effective deadtime of a μs .

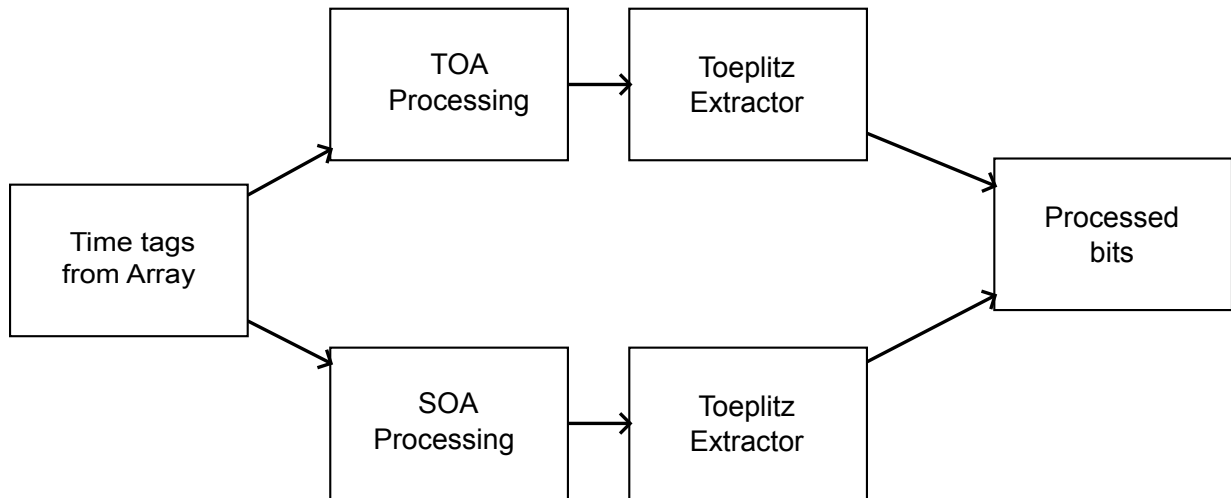


Figure 2.14: Block diagram showing schema for processing data in parallel. The basis for a potential spatiotemporal QRNG.

Chapter 3

High-Speed Streaming Software

In this chapter the development of the streaming software is described. The chapter starts with a discussion on the QEYSSat requirements as they relate to the streaming software and a bit of background on the AWG. Next, the conversion files that are responsible for initially transforming the unbiased bits into WCP states and from there to AWG samples are introduced. After that, the first iteration of the streaming software is presented, followed by analysis of its shortcomings. These deficiencies are subsequently addressed in the second major iteration of the software.

3.1 QEYSSat Requirement

The QEYSSat missions proposed WCPS requires an arbitrary waveform generator (AWG) with two key requirements, listed below.

- 4 independent but synchronized channels.
- Streaming capability, where the unit outputs a waveform on the fly that is specified by an external source.

Where the streaming capability arises from the amount of memory that would be required to output an entire passes worth of random states. Recall earlier that the longest satellite pass was assumed to be 600 seconds. With a 400 MHz source rate, and assuming 2 bytes per pulse (where each channel is needed per pulse) gives 3.2 GB/s over 600 seconds yielding a total memory requirement of 1.92 TB. Downloading and outputting over a TB of data is

unfeasible, and so the need to be able to stream data on the fly at the previously mentioned rate of 3.2 GB/s is necessary. These two facets are uncommon in most AWG's, and the only suitable commercial alternative found at the time of this writing would be to operate 2 Keysight M18121A model AWG's in parallel with one another. Another option would be to develop a custom design FPGA, but that was not considered in greater detail due to the risk associated with a longer development time-line.

3.2 Arbitrary Waveform Generator Overview

In order to readily follow the discussion on the streaming software, it is important to go into greater detail regarding one of the most important pieces of hardware for the WCPS system utilized in QEYSSat. That is the AWG which is responsible for sending electrical signals to specific components of the WCPS. This section will focus on a description of how the AWG operates, as well as an introduction to the AWG chosen for the WCPS, Tabor's P2584M model.

3.2.1 Arbitrary Waveform Generator

Without diving into the specifics of signal generation, a straightforward way to understand an AWG is by thinking of a simple sinusoidal analog signal. Now take that signal and consider every single infinitesimal point that it is comprised of. To characterize each point assign a number along the time axis to it as an x coordinate and a potential as a y coordinate. We now have a sequence of unique tuples frozen in time that each have a voltage associated to them. If now that sequence of points is produced in the same order that was just determined previously, the original signal would be produced. This idea of breaking a signal down into points is the basis of an AWG[67].

Realizing this concept introduces several electrical components that facilitate production of these sample points. As a voltage cannot be produced with infinite precision, one limitation that quickly arises is the amount of information associated with each sample point. Since an AWG is comprised of complex hardware, this amount of information can be thought of as how many bits are used to describe the voltage of the waveform. These discrete bits comprise a digital signal that the AWG must convert to a continuous analog one. This conversion of discrete bits (per sample) to a continuous signal is done through the use of a digital-to-analog converter (DAC). The precision of the DAC is directly related to its bit-depth. A 4 bit DAC can only output 16 discrete levels. Think of this as dividing

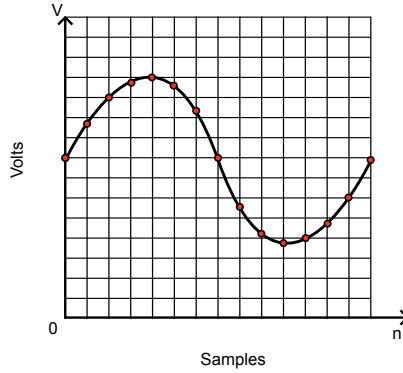


Figure 3.1: AWG breakdown via sine wave into points and voltages

the y-axis on [Figure 3.1](#) into 16 equal sized steps. Where the minimum value is zero and the maximum value is some max voltage as specified by the hardware. The operational speed, as far as the output is concerned, is set by the devices clock rate. Operating at 1 MS/s the AWG will output 1 sample point every microsecond.

Tabor P2584M

The AWG chosen for the QEYSSat mission is the P2584M model AWG from Tabor electronics. This device, pictured in [Figure 3.2](#), comes equipped with the two previously described key features. The P2584M defaults to operating with a 16 bit depth, though 8-bit samples can be used with the added constraint of a loss in precision[\[48\]](#). As for the streaming mode, each channel gets its own 32 MiB buffer that is further broken down as 16 1 MiS sections. The unit itself can be thought of as having 2 parts, each part has its own memory and DAC and 2 channels are assigned to each part[\[48\]](#). Both parts operate on the same system clock. Together, this constitutes the creation of inter and intra part dependencies. For streaming, the main inter-part dependency is that each channel outputs at a rate designated by the sample clock rate[\[49\]](#). This avoids having to monitor each channel individually and synchronization between the channels is maintained by default. This is only the case if the computer responsible for providing the streaming data does so at the required rate.



Figure 3.2: Tabor P2584M model housed in its chassis.

3.3 Streaming Overview

3.3.1 QEYSSat WCPS

From the AWG point of view, the WCPS is comprised of four blocks as shown in [Figure 3.3](#) in addition to receiving data from the control computer. They are the laser trigger, intensity modulator, synchronization, and polarization modulation. Ergo each WCP state to be produced is defined by

1. Whether a non-vacuum state is being sent (in accordance with the BB84+ protocol)
- laser trigger block
2. Whether the state is signal or decoy - intensity modulation block
3. Where we are in the non-repeating sequence - synchronization block
4. the polarization of the state - polarization modulation block

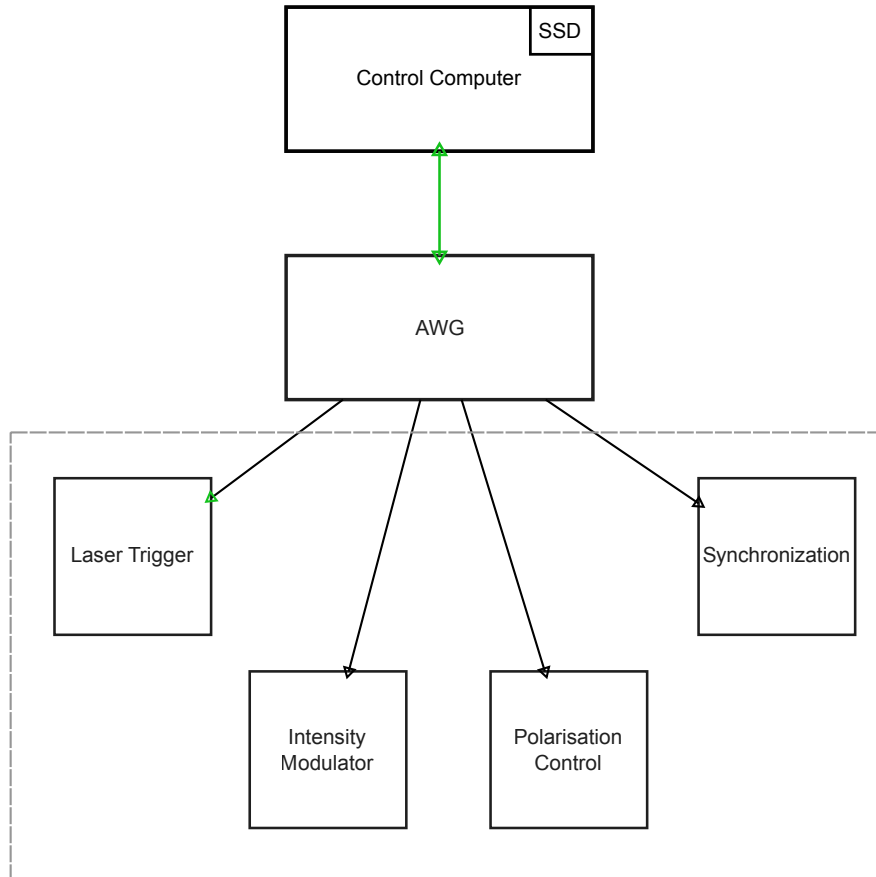


Figure 3.3: A block diagram representing the QEYSSat WCPS as viewed from the AWG. There are four distinct components, each one having its own dedicated channel from the AWG. Communication between the AWG and control PC is done via a eight lane generation three PCIE. The data file containing the AWG samples to be streamed is stored on a fast SSD. This is to ensure the data can be read from file efficiently to avoid inhibiting the streaming software.

These four blocks are then partially realized as voltage levels specified with a precision limited by the AWG hardware. A single channel of the AWG is dedicated to a particular block of the WCPs. From there the problem shifts solely to addressing each channel of the AWG with the correct waveform data at the correct time. This means every WCP state must first be broken down into its constituent parts, each of which is subsequently transformed into waveform data to eventually be transmitted to its respective block. There are two configurations one might readily intuit for the format of the waveform data. The first being each WCP state component remains attached to each other, and sequential WCP states are then placed one behind the next. This would then lend itself, from the AWG's perspective, as having the individual channel data interleaved in the overall waveform data. The second, is that each WCP component is completely disassociated with its relatives. This means that all the waveform data associated with the laser trigger block is stuck together. Of course, the other blocks of waveform data would then also be stuck to one another. From the AWG's perspective, this would mean grouping the data for each channel together. In [Figure 3.4](#), the difference between the two formats is examined visually.

While so far the choice of data format has been presented as subjective, in reality the manufacture specifications for the AWG dictated a particular format. Before examining this facet of the streaming software, I first want to shift the spotlight briefly onto the various processes that must take place prior to streaming. The streaming software is concerned with waveform data, which is derived from random WCP states ultimately formed by the QRNG discussed in the previous chapter. That means first the random bits output from the QRNG must be transformed into random WCP states, which are then subsequently transformed into AWG samples. The result of each transformation is kept in a file. The file containing the AWG samples is the main input into the streaming software which will ensure the correct data is sent to the correct channel. The scripts that are responsible for each transformation were originally authored by Dr. Brendon L Higgins. The script responsible for transforming the WCP states into AWG samples was modified by myself to match the input data format expected by the P2584M model. A brief description of the script responsible for transforming the raw unbiased bits into WCP states is given below:

1. Input to the program two files. One contains the unbiased random bits, the other contains parameters associated with the BB84+ protocol (e.g signal-decoy-vacuum probability)
2. Read unbiased random bits from file.
3. Generate biased bits dependant on signal-decoy-vacuum probabilities from unbiased input.

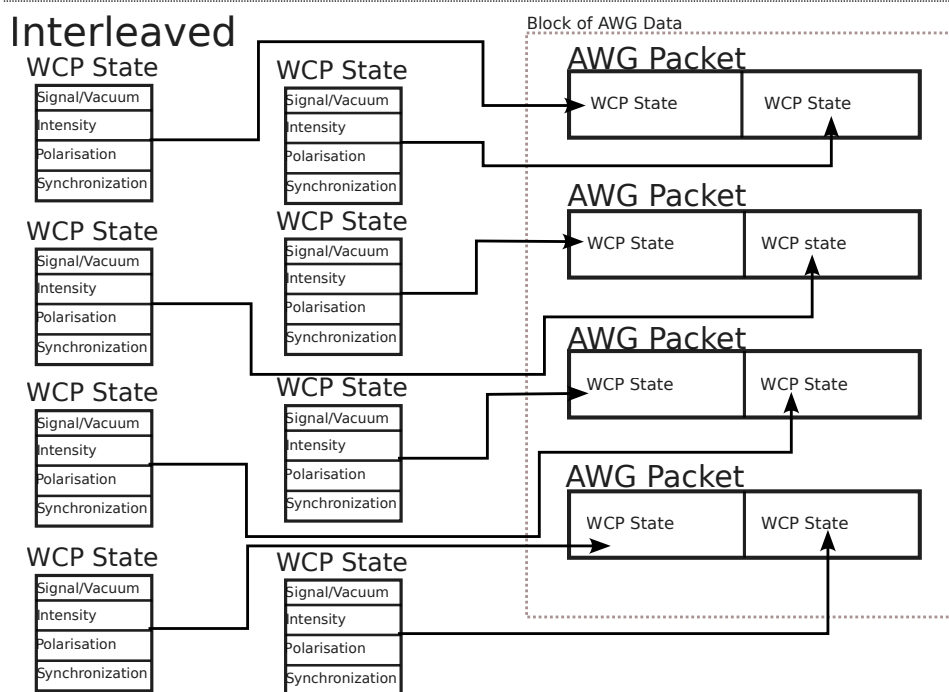
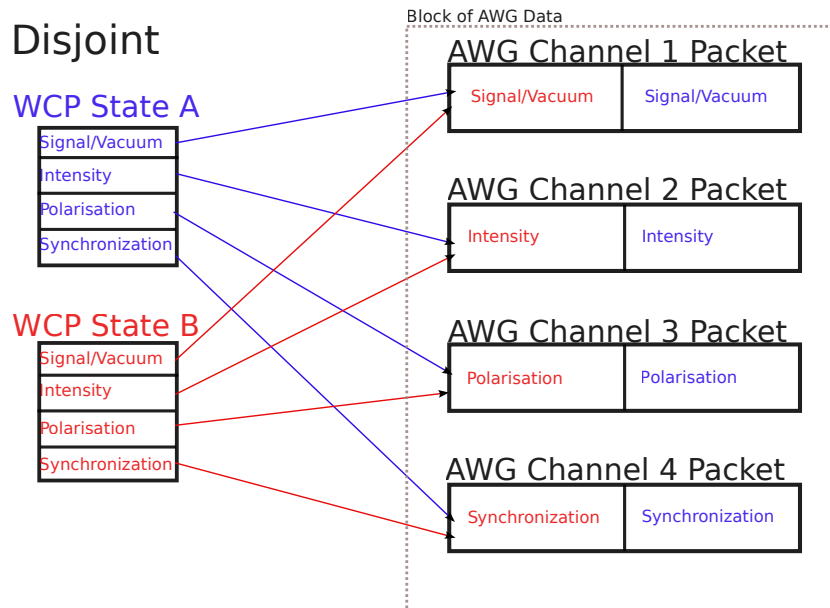


Figure 3.4: Interleaved versus disjoint data formats

4. Assign polarization to each WCP state
5. Output to file

It should be noted that there is a particular encoding to the output WCP states, which for the purposes of understanding the WCP states to AWG samples script will be simplified to only be that each WCP sample is encoded in half a byte. With this in mind, I will now return to examining the data format of the AWG samples file which will then lead into a brief explanation of the second conversion script.

The QEYSSat mission baseline is to operate at a 400 MHz source rate. In terms of the AWG this would translate to a 400 MS/s rate for each of the four channels operating with the default 16-bit depth. This 3.2 GB/s streaming throughput is the minimum rate the streaming software must be able to reliably operate at for the duration of the satellite pass.

3.3.2 Streaming File Format

It turns out the P2584M model, perhaps unsurprisingly, assumes the disjoint data format[\[49\]](#). On top of this, the model also has a predefined packet size of 1 MiS for streaming data. Combined with the knowledge that each sample is 16 bits or 2 bytes I may also refer to the packet size of the AWG as 2 MiB. Since each WCP state is spread across all four channels of the AWG, it is convenient to group one packet of channel data, for each channel, as a block of WCP data. The file format for the AWG samples can then be viewed as simply a header, followed by blocks of WCP data where each block of data is further deconstructed as four individual packets. Each packet in a block will then be for its respective channel. As for the ordering of the individual packets, this is largely subjective to which WCPS component each channel of the AWG talks to.

Now, as for the WCP state to AWG samples script is concerned, the main modification I made was to adhere to the above ordering of packets in a block of WCP data. The steps the script performs are encapsulated in [Figure 3.5](#).

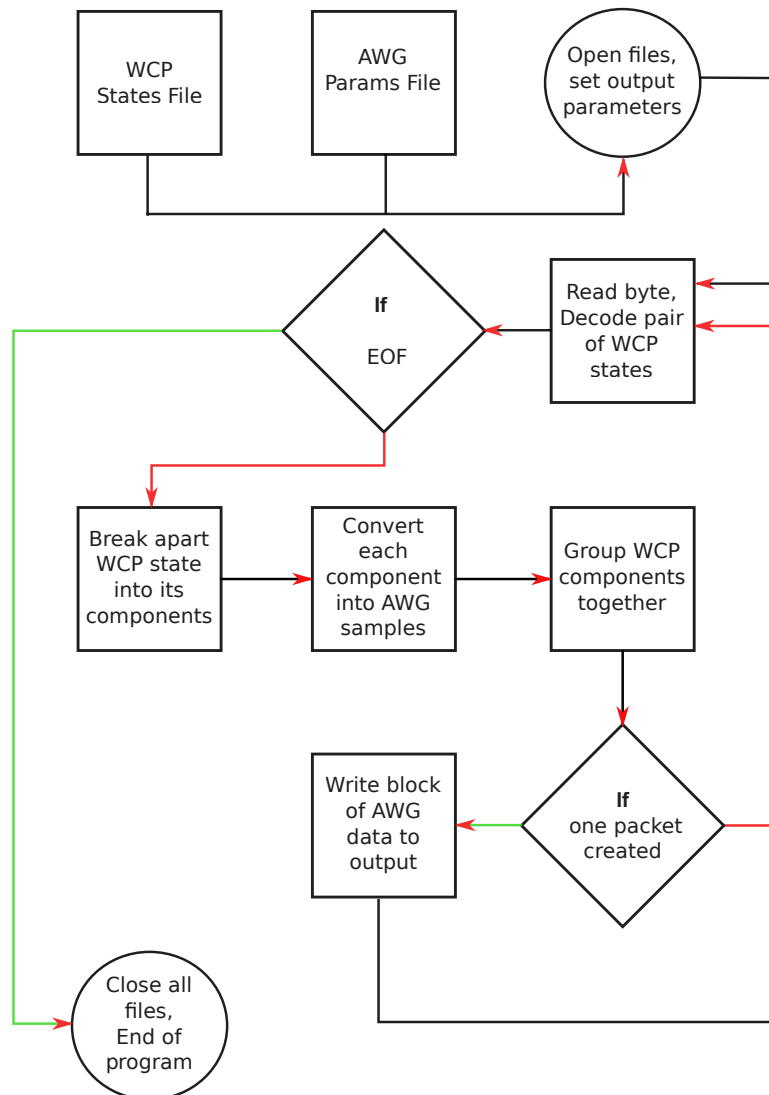


Figure 3.5: A flow diagram representing the general steps undergone in the second conversion script responsible for turning WCP states into AWG samples. The changes I made to this script would be encapsulated in the "group WCP components together" block. Here each WCP state would be broken into a polarization, synchronization, intensity, and trigger component. These individual components would then be stored temporarily in an array with a size equal to the AWG packet size. Once a packet of samples was made all four arrays of AWG samples would then be written to the output file.

3.4 Streaming Software Implementation

Since the AWG data is stored in a file, two overarching steps must be completed before sending data to the AWG. Firstly, the data must be read from the file into a buffer. Next, that buffer - which at this point can be viewed as a packet - is then pushed to the AWG. Both of these operations then repeat until a user-defined condition is met, which in this case is reaching the end of the file.

3.4.1 Streaming Software Iteration 1

The first version of the streaming software approached the problem by viewing it as a Producer-Consumer problem[19]. The producer was responsible for reading the streaming data from the file into a empty buffer to fill it, and the consumer was responsible for streaming full buffers of data to the AWG. 8 MiB buffers were used, where each buffer was broken down into four distinct smaller buffers of size 2 MiB. Every group of four buffers was managed by an abstraction class termed a "buffer manager". Altogether this collection of 8 buffer managers comprise a shared resource that formed the basis of the Producer-Consumer implementation. Access to each buffer manager was mediated through the use of two semaphores¹, full and empty, that kept track of the number of full and empty buffers respectively. Figure 3.6 is a diagram that demonstrates the software execution.

Care was taken to avoid the usual pitfalls of multi threaded programs, like race conditions and thread deadlocks[31]. Race conditions occur when access to the shared resource is a free for all and ultimately up to the operating system and subject to change each time the software is run. Avoiding a race condition is important as this could have the result of transmitting WCP data out of order. This would ruin the reconciliation phase of the QKD protocol as Alice would not have sent the states in the intended order. Ultimately, this would lead to an increased error rate and increased likelihood of failing to generate a key. As for thread deadlock, this occurs when each thread is waiting for the other to do something before continuing. Both threads halt and in the case of only two threads the program stalls indefinitely.

¹A semaphore is a type of synchronization primitive that mediates access to a resource[17]

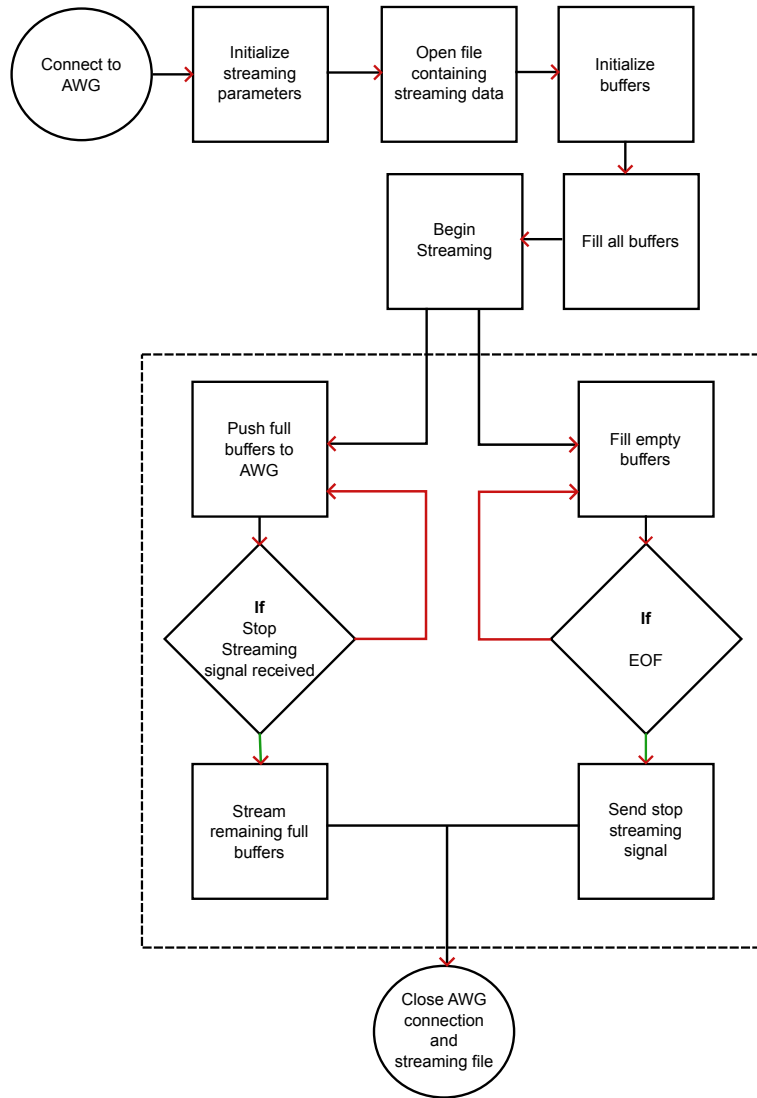


Figure 3.6: The above flow diagram is for the streaming software's first iteration. The dashed box shows where the producer-consumer problem is being implemented. The left side would correspond to the consumer or stream thread. The right hand side is the producer or read thread. The read thread is responsible for breaking the program out of this multi threaded loop by notifying the other thread once the end of file flag has been encountered. Then all remaining packets/full buffers are sent to the AWG and the two threads are joined to subsequently end the program by disconnecting from the AWG. If the program encounters an error when pushing packets or reading the file streaming will halt and an error message will be output to console.

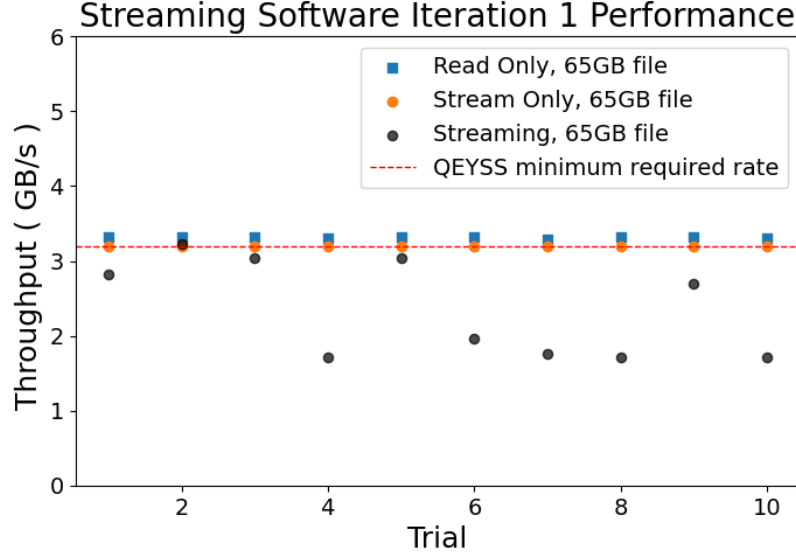


Figure 3.7: A 65.536 GB file was created containing waveform data to be streamed to the AWG. Each of the three data sets have a related but slightly different meaning to the streaming software. The read only values refers to running the streaming software without sending packets to the AWG. This is a good indication of the IO performance of the program. The stream only values are the inverse, fake data was made once and sent repeatedly to each channel. Streaming corresponds to running the program normally. The ideal scenario would be to have the streaming data points match the required minimum rate with minimal fluctuation.

Results

The performance of the first iteration of the streaming software is summarized in [Figure 3.7](#).

The main takeaway from the above figure is that while the system - the controlling computer and AWG - is capable of sending packets at the required rate, as demonstrated by the stream only results, the program itself fails to do so. It should also be noted that the streaming was specifically configured for the baseline parameters of the QEYSSat mission, corresponding to a throughput of 3.2 GB/s. In only 1 of the 10 trials could the throughput be considered a pass. However, a value above the required rate is potentially misleading because the rate demanded by the AWG should never exceed 3.2 GB/s. Meaning, the throughput reported above should rarely be noticeably higher than the set rate. This would

lead to questioning whether all the packets contained in the file were actually successfully streamed to the AWG. Additionally, it is important that the streaming software be reliable and any performance fluctuations be minimized. [Figure 3.7](#) makes it clear the streaming not only is typically under performing, but also very unstable.

For the remaining trials, the question remains what caused such a huge degradation of performance. The answer can be found via the idea of a sustained minimum value. All the values for the read only thread correspond to averages. The real-time IO performance can fluctuate, sometimes drastically to the point of the AWG throwing an error due to insufficient resources[49] (the AWG wanted data to output and there was not any data available). There are numerous reasons for a disk's read speed to fluctuate, for example device age and amount of free storage can impact the read speed[47]. There's also caches to consider. If you were to have two separate functions that each opened the same file and read it from start to finish and times both of them the one that accesses the file second would be noticeably faster as a larger portion of the data would be cached locally. This is somewhat mitigated by having a test file that is several tens of gigabytes large. Furthermore, there are no guarantees that no other processes are accessing the disk while the streaming is occurring and this would impact the IO speed. Altogether, the need to improve the read thread's performance to have a sufficient and stable margin over the streaming thread is necessary.

For QEYSSat, the overall performance of the first iteration of the streaming software is insufficient as it does not consistently and reliably stream data to the AWG at 3.2 GB/s (corresponding to a 400 MHz repetition rate for the WCPS). While the consumer is capable of providing the AWG full buffers, the producer cannot regularly produce full buffers fast enough. This leads to a failure of the software to maintain a sustained minimum throughput in excess of the baseline required rate that is necessary for the AWG - and by extension WCPS - to operate correctly.

Open Question

When looking at the results in [Figure 3.7](#) it is clear that the current methods used by the read thread to parse the data file need to be revamped. Additionally, the read thread performance is ultimately limited by the performance of the drive that is being read from. However, as [Figure 3.8](#) shows, when tested, the sequential read speed of the drive being used should far exceed the requirements of QEYSSat. The testes drives sequential read speed is 1.75 times greater than the QEYSSat requirement of 3.2 GB/s. So, then the question arises how was the result in [Figure 3.8](#) obtained?

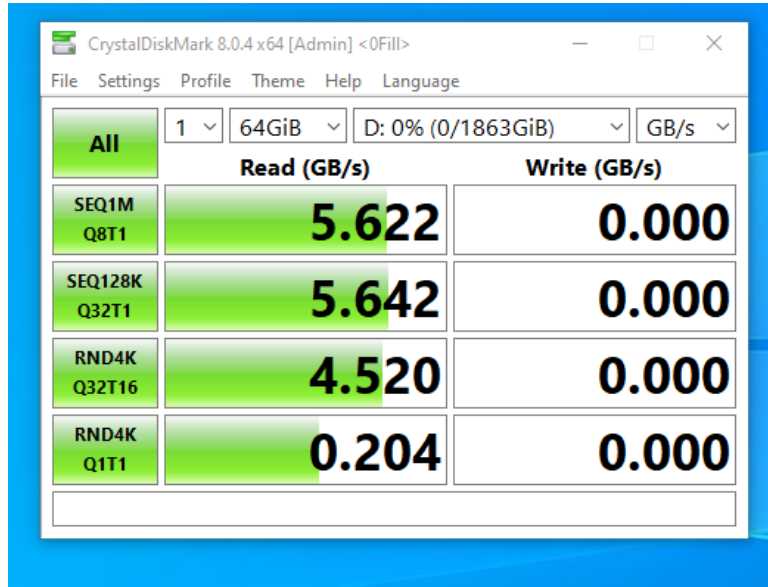


Figure 3.8: Result of running Crystal Disk software[46] on the control computer. The most meaningful value in the image is the sequential read speed value, as this corresponds to reading the streaming file from beginning to end continuously (no jumps).

3.4.2 Streaming Software Iteration 2

The previous section makes it clear that in order to reach the streaming requirement for QEYSSat (3.2 GB/s) the read thread must be improved. An investigation into Crystal Disk’s repository led to another repository from Microsoft, *diskspd*, that was used to actually measure the disks performance[40]. Subsequently investigating how *diskspd* works led to careful refactoring of the streaming software to take advantage of more sophisticated IO routines.

IO operations

For many programs, utilizing a programming languages abstraction of a read method is sufficient. Little to no thought is given to what the operations associated with opening and reading a file are actually doing. Most of the time, the default behavior of these methods readily handles the requirements of the user. For example, calling a programming languages open file function can be oversimplified to simply being asking the underlying operating system to find the requested file and create a connection between the program

and that particular file. There are various ways to specify how you want to open a file, for what type of IO are you opening the file, the typical mode of accessing the file, security, and miscellaneous other attributes[45]. For the streaming software, the most important specifications are the type of IO, the mode of accessing the file, and one miscellaneous attribute related to IO buffering. Breaking down these further, the two main types of IO are synchronous and asynchronous. Similarly, there are two main types of access, random and sequential. For buffering, the situation is a bit more complex but can essentially be broken down into no buffering or buffered. Each of these will be examined in greater detail in subsequent sections.

Also, it should be noted that while the underlying operating system has a definite impact, I will do my best to avoid most of the details related to implementing the different types of IO for a specific operating system. Instead, the focus will be on the concepts and how they impact the performance of this piece of software. That being said, since I dealt with a Windows operating system most of the sources I will refer to are from Windows documentation.

Synchronous IO

As the default behavior of IO operations, synchronous IO has one key defining characteristic. That is that the calling thread immediately waits for the read operation to be completed before continuing. This is due to the interface between the process and the core of the operating system, the kernel. In order for a process to obtain the data from the file that is opened, it must continue to use the kernel as the intermediary. The kernel is then responsible for retrieving the requested data from the designated piece of hardware[2].

While the kernel is busy executing the relevant tasks to complete the IO operation, the calling thread is doing absolutely nothing. From an efficiency standpoint, this is sub optimal for a multitude of reasons. Firstly, from the threads point of view, it could be busy executing other tasks, or performing other computations whilst the kernel is busy with the IO request. Secondly, from the drives perspective, it has to seek and retrieve the requested data for each request. For synchronous IO this entire process executes and then the drive stops 'working'. Every subsequent synchronous IO request then involves stopping and starting the drive - regardless of the type SSD or HDD. Though for an HDD it is easier to visualize since the actual disk head has to spin to the designated area - then begin reading the contents - and then transmit the read data back to the kernel. This type of inefficiency is somewhat mitigated through the use of cache and pre-fetch[1].

Caches are temporary data storage that are significantly faster than other types of mem-

ory storage (i.e like that found in a hard drive). Cache's contain a copy of the relevant data and are checked before going to deeper into the hardware level and having the relatively slow drive fetch more information[9]. Many systems employ multiple cache levels, which are typically labeled 1 to N. Level 1, or L1, would correspond to the smallest but fastest cache. N would be the slowest but biggest. In the case of three distinct cache levels, they will be termed L1 - L3. To compare speeds, an L1 cache will typically be able to support a bandwidth of 1TB[9]. An L3 cache will have a bandwidth of several hundred GB[9]. Compared above with the performance of a fast SSD of 5 GB/s the difference in speeds is notable. A typical IO operation (or any kind of memory access for that matter) will first check the differing levels of cache for the requested information. This is the subject of cache friendly code, or cache coherency. Its main relation to IO is the use of pre-fetch[14]. Essentially, while accessing memory stored on a disk various algorithms are employed that seek to predict - and do so with high accuracy - what information the process will likely also request. For reading a file, an easy way to visualize this is that instead of reading only 1 line of the file, the disk will instead read 3, and the information from the next 2 lines of the file will be cached. This is where the type of access for the file comes into play. By specifying that a file will be accessed sequentially, the amount of data that can be aggressively pre-fetched and subsequently cached increases[45][14]. Thereby reducing the number of times the underlying drive has to spin up and retrieve the requested information to complete a particular IO operation. If instead the file access was specified to be done randomly, a different algorithm would be employed.

On top of cache's IO operations are typically buffered at the system level[3]. This is related to the idea of pre-fetch in the sense that the system has its own buffers, as does the user. Essentially, the data coming directly from the disk is first brought to a system buffer, and then to the user buffer. The intermediary step of using the system buffer is not dissimilar from a cache - in the case of Windows it is actually a part of the file caching process[1]. For large reads of a file, the cache can become cumbersome as the amount of data requested exceeds the size of the cache. Therefore, not only is the disks overall movement inhibited, the data the disk does retrieve is first cached once before being sent to the user buffer. Ergo, the data is copied twice (once to cache, once to the user) each time this cycle occurs. As hundred of thousands of request's are made this will negatively impact drive performance and overall performance of the overarching process. This inefficient caching can be disabled in Windows through the use of the `FILE_FLAG_NO_BUFFERING` flag when opening the file[45]. However, additional constraints are imposed on the users program to ensure proper operation. These constraints can be succinctly summarized in the following way: all file access sizes must conform to disk characteristics. For the case of Windows, this is encompassed through allocating memory in a particular fashion[42],

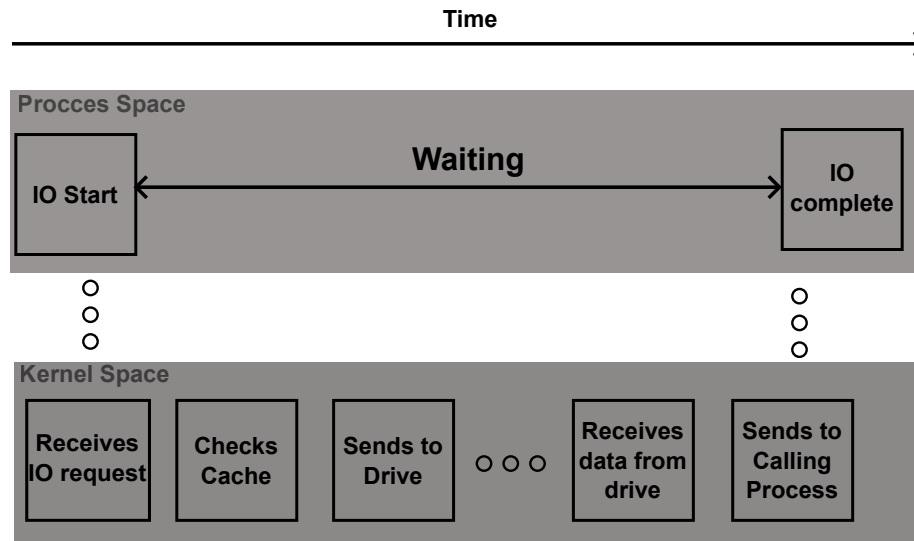


Figure 3.9: An example of the general process of a synchronous IO call. Note that the calling thread is immediately put into a wait state after issuing the IO request. All the successive steps involved in completing an IO request from the kernel side - such as checking cache's - are done independently of the thread.

proper use of the overlapped structure^[43], and obtaining the disk geometry via the correct windows API such as `IOCTRL_DISK_GET_DRIVE_GEOMETRY`.

Asynchronous IO

Performing IO asynchronously is a more complex endeavor, as the calling thread is no longer forced to wait for the IO operation to be completed. Therefore, the thread is free to perform other operations. However, at the same time mechanisms must be integrated to help keep track of each asynchronous IO request^[42]^[40]. Furthermore, each IO request must have a unique buffer to store the retrieved data after the request has been fulfilled. Additionally, these buffers must remain available and unmodified throughout the duration of the request. The benefit of asynchronous IO is that the kernel and user process can continue to operate pseudo-independently of each other. Therefore, the kernel can now keep the underlying drive busy and mitigate the losses associated with the stop-start cycle associated with synchronous IO. This is especially true in the case of direct IO in conjunction with modern SSDs. In many cases, the underlying hardware is more than capable of exceptional performance but finds itself handicapped to a degree by the operating

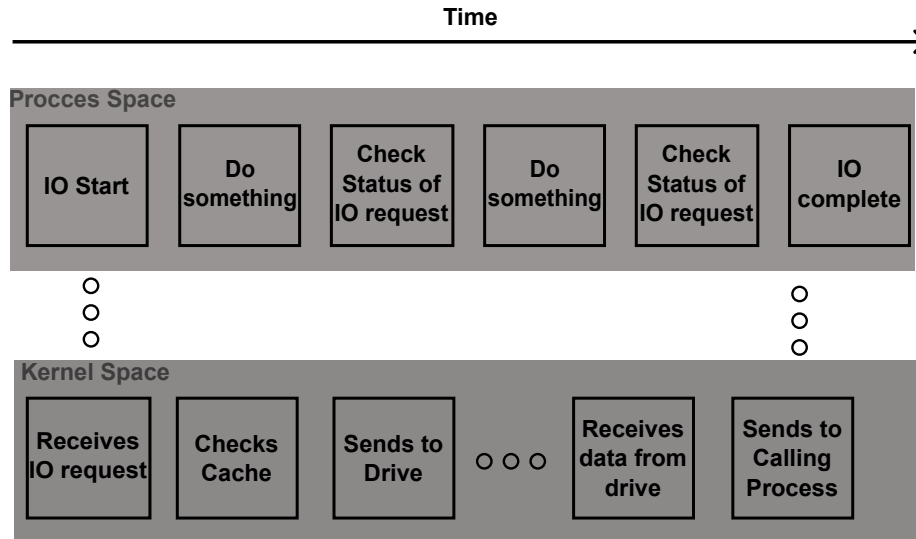


Figure 3.10: The only difference between synchronous and asynchronous IO is now the calling thread is free to do other activities, which in the case of QEYSSat amount to attempting to issue more IO request. There is additional complexity introduced by the need to keep track of each IO call and the resources associated with it. Apart from that, the steps involved are the same but by being able to issue more IO calls while the previous ones are being completed the overall read throughput increases substantially.

system[54]. As for the most efficient set up, that is more of a test-and-check situation with a particular drive as each drive will have its own sweet spot in regards to concurrent IO requests and performance[33]. Lastly, a potential area for improvement is via thread priority. Each thread is assigned a priority, and higher priority threads get access to resources first. In the case of Windows, this is broken down further between a process priority class, and the threads priority class[10]. The only caveat is that when using the highest priority classes, it is possible to get the operating system to ignore other very critical processes like peripheral inputs. So, in almost every scenario it is correct to not use the highest priority class unless interfacing directly with the hardware for brief periods of time.

Streaming Software Iteration 2 Implementation

From the first to the second iteration of the streaming software, the main change is switching from synchronous to asynchronous IO. This involved a non-negligible refactoring of the software to accommodate OS specific calls and structures. Additionally, direct IO was

used in conjunction with specifying sequential access. While specifying a sequential scan of the file in conjunction with direct IO would appear to be mutually exclusive as the user process is in charge of the buffering, there are still cached process that may be impacted by specifying sequential access[3][14]. The abstracted buffer manager class was also adjusted to include an event handle to be associated with the asynchronous IO request, and four individual buffers were consolidated into one large buffer whose size was increased to be 2 blocks or 16 MiB. Now the portions of data corresponding to each channel of the AWG are maintained by pointers to specific offsets in the buffer. Only four pointers are used, and the pointers are incremented and reset as needed. This also means that the IO request sizes have doubled from the previous implementation, and the number of read calls have decreased by a factor of 8. In summary:

1. Synchronous IO $\longrightarrow$ Asynchronous IO
2. Buffered IO $\longrightarrow$ Direct IO
3. 8 MiB buffer (1 packet per channel) $\longrightarrow$ 16 MiB buffer (2 packets per channel)
4. Refactoring buffer manager class

As far as the software execution, the read thread now primarily submits IO requests to the kernel. The stream thread now checks the status of the request, and if unfulfilled with no errors waits for it to be completed. This is to ensure the packets are streamed in-order as there is no guarantee as to the order the kernel will choose to fulfill the IO requests. Before entering the producer-consumer set up, an initial burst of IO requests is made to the kernel - with the same goal to initially start in a scenario where the buffers are primed and ready to be streamed immediately.

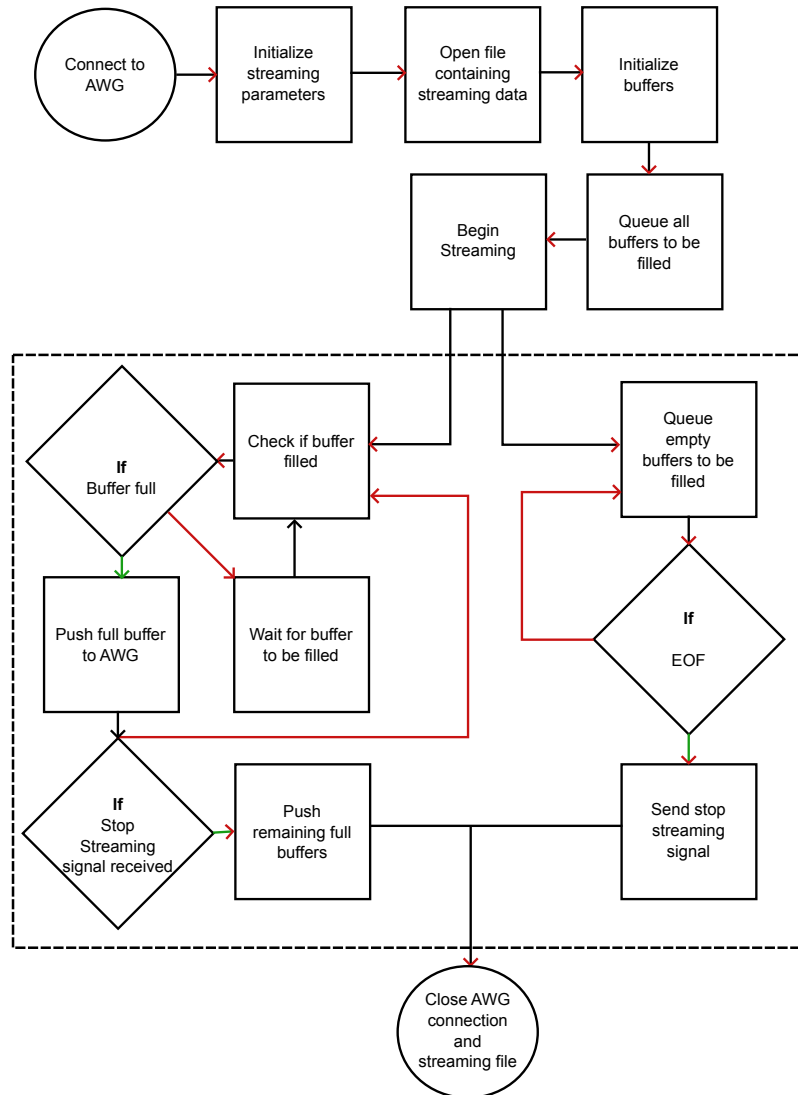


Figure 3.11: As in the previous section, this flow chart represents an overview of the software’s execution steps. The increased complications from using asynchronous IO can be seen visually with the changes in the number of steps each thread is completing. Furthermore, the initial filling of the buffers is now instead a burst of IO requests for the system. The goal is then to sustain a finite amount of IO requests to ensure the drive works continuously.

Results

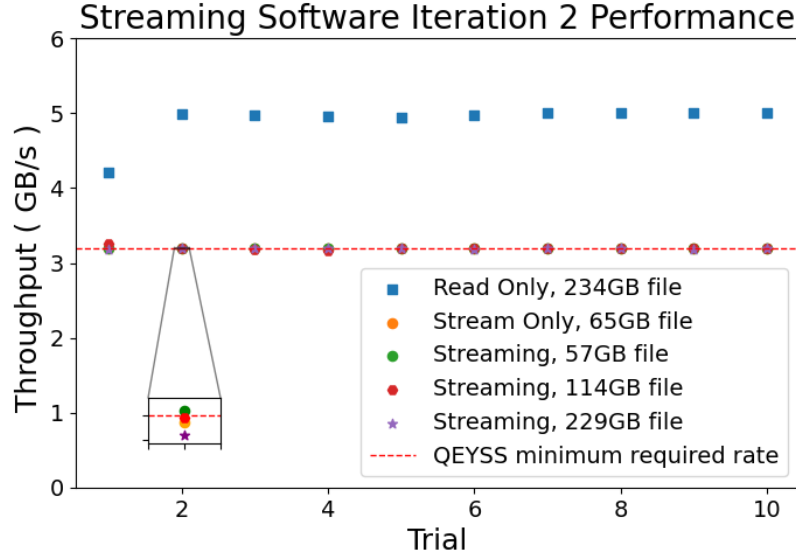


Figure 3.12: As before read only and stream only refer to a particular mode of operation for the streaming software. In this case, various file sizes were used to test how the streaming software behaves when operating for longer periods of time.

The results of [Figure 3.12](#) show a clear improvement to the streaming software performance and stability. The longest satellite pass times correspond to a file size of ≈ 200 GB. The last column in the table shows the ability of the streaming software to reliably meet this target. That being said, the first trial for each file size was, for the larger file sizes, notably different than its peers. This could be from a multitude of parameters, from variations in the number of outstanding IO requests[\[33\]](#) to the nature of IO operations itself. As the software is still multi-threaded the read thread is relegated to simply issuing IO requests while the streaming thread handles the decision making depending on the status of each IO request. There could be some inefficiencies there, since IO is ultimately a serialized operation which could impede a multi threaded approach. This is the most likely case as once the threads are consolidated, streaming a 234 GB file 5 times - see [Table 3.1](#) - showed very little fluctuation in comparison to these results.

Reevaluation of software implementation archetype

The producer-consumer problem archetype that was applied to create the streaming software is more suited to a scenario that uses a working buffer[19]. In the case of the streaming software each buffer is either full and ready to be pushed to the AWG, or empty and being handed off to the kernel as part of an asynchronous IO request. Furthermore, the read thread is now significantly faster than the stream thread. So much so, that the read thread is predominately idle while it waits for the empty semaphore to be incremented. Additionally, the streaming thread will wait for the IO request to be completed before pushing any packets. Previously, it was stated that the "buffer" was effectively the sum total of each buffer manager's buffer. Then the producer can be thought of as filling a particular amount of the buffer and the consumer empties a particular amount of the buffer. Each of these processes occurs at a different rate. The producer's rate is ultimately limited by the drive performance, which is now operating around the manufacturer specified maximum rate of 5.0 GB/s. The consumer is operating at a rate specified by the AWG, which as a baseline will be 3.2 GB/s. Each process ends when the file has been read entirely. With these parameters in mind, this situation can be viewed as draining a reservoir with the help of a smaller tank. Filling up the tank is the job of the producer, and emptying the tank is the job of the consumer. The tank is the buffer. Once the reservoir is depleted, the problem is solved. As seen in [Figure 3.13](#), the tank is almost always full, and any time the consumer empties a portion of the tank it is immediately filled by the producer. Furthermore, the producer is eventually forced to operate at the same rate as the consumer. Altogether, the need for the tank is dubious at best, and it turns out that it can be removed. In other words, the producer-consumer problem is no longer applicable, and its original application to an IO bounded problem was incorrect. The last major change to the streaming software was to consolidate the producer and consumer threads, and as shown in [Table 3.1](#) the results are effectively the same as before with the added bonus of reducing the software's complexity.

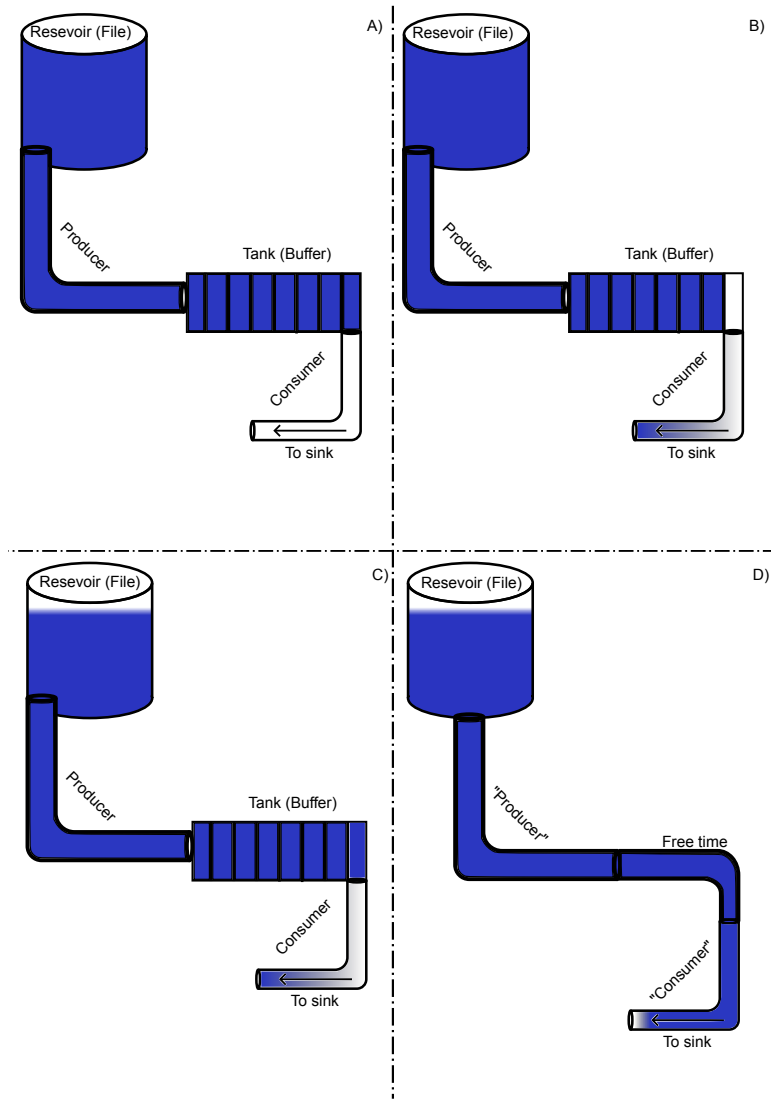


Figure 3.13: A diagram explaining the above metaphor of a reservoir and a tank. Water from the reservoir (or data from the file) is taken and temporarily stored in a tank (labeled as buffer in the diagram) with 8 subdivisions - one for each buffer manager. When one managers buffer is sent to the AWG, it is immediately queued up to be refilled, and the overall tank never falls below 87% full. As this process continues to repeat endlessly until the reservoir there is no real need for a tank and instead the reservoir could be drained in one continuous path without a tank. Due to the difference in pipe sizes (speed of the read versus stream thread) eventually the entire process runs at the slower rate. This can actually be directly observed by observing the drive IO rate via a resource manager. Initially the IO speed is very high, but eventually it matches the streaming thread.

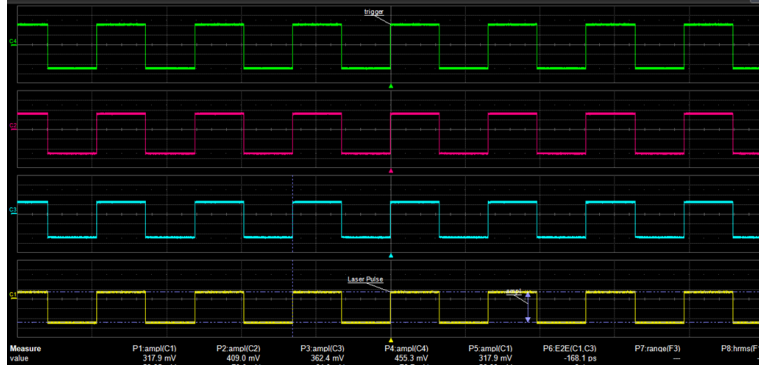


Figure 3.14: Streaming software output example. Corresponds to running the the second iteration of the streaming software. Captured with an oscilloscope.

Streaming
3.199
3.206
3.206
3.206
3.206

Table 3.1: A small experiment where a 234 GB test file was streamed after consolidating the threads to ensure streaming software functioned as expected. Corresponds to the above image of the streaming output. Meant to verify the streaming operates as expected post consolidation. Thread consolidation has not only produced results that are consistent with the previous multi-threaded approach but also improved the software by decreasing complexity and improving stability.

3.5 Outlook

Iteration one of the streaming software was based off of modeling the problem as a producer-consumer type. The read thread (producer) was responsible for parsing the data file quickly enough for the streaming thread (consumer) to send to the AWG. However, the average performance of the read thread hovered just above the required rate for streaming. This implied that the minimum rate of the producer would fall below the specified rate. The result of this shortcoming translated to erratic behavior of the AWG output that is

succinctly captured in the variability of streaming rates visible in [Figure 3.7](#), provided the device did not throw an error.

Iteration two addressed this by changing the parameters involved for reading the file. The use of asynchronous and direct IO substantially improved the read thread performance as well as the overall streaming performance. Now, the streaming software was able to run consistently and reliably at the baseline rate required by QEYSSat, 3.2 GB/s, which was the ultimate goal of this project as shown in [Figure 3.12](#). Additionally, the software complexity was able to be reduced by reevaluating the need for a producer-consumer implementation. Applying the producer-consumer model to this problem was incorrect for one main reason. The streaming software is ultimately an IO bounded problem. Using multiple buffers, threads, and constructs cannot overcome the IO performance of the software by itself. It can mitigate any minor variations in the read speed by having other buffers ready to be used but that solves nothing if all the buffers end up empty after a few seconds of operation. At that point the software is basically using one buffer, which ended up being the case for the first iteration.

To keep the software performing correctly, care should be taken to keep the SSD drive used relatively empty as IO performance degrades as less memory is available[\[47\]](#). As a preliminary to attempting to drive the WCPs at faster rates, the sample rate of the AWG was increased and the streaming performance evaluated. A performance degradation in the streaming was observed around 480 MS/s when operating the streaming software in stream only mode. This would correspond to a throughput around 3.8 GB/s. As the connection between the controlling PC and the AWG is a 8 lane gen 3 PCIE capable of operating at rates up to 7.88 GB/s[\[50\]](#), the data transfer rate did not appear to be the problem. Additionally, as the data was pre-made fake data living in the processes RAM - which has transfer speeds exceeding 10 GB/s, that would also appear to not be the problem. Therefore, it is my opinion that the problem is more subtle and deals with the drivers of the AWG itself. This question was raised to the manufacturer and the investigation is ongoing. However, as for the baseline requirements for QEYSSat are concerned, this is not a problem and the streaming software should work as expected.

A brief note on data storage. It was previously mentioned that over a terabit of random bits are needed to generate all the WCP states. As the WCP states are stored in each nibble of a byte, that data file has a size of $\approx 500Gb$. The streaming states are all expressed with 16 bit precision for each of the four components of a WCP state implying file sizes over a terabyte. Storing the streaming data file for a year, assuming at least 1 pass per day would take at least 365 TB. However, any particular streaming file can be recovered from the conversion script responsible for transforming WCP states to AWG samples. Therefore, the only file that is absolutely needed to be kept long term is the WCP states definitions

file. This would greatly lower the amount of data storage associated with QEYSSat. Of course, any kind of storage should not involve the SSD used to stream the data for the reasons described in the preceding paragraph.

Apart from addressing the previously mentioned throughput cap of ≈ 3.8 GB/s, another avenue to pursue would be working with the manufacturer to implement a jump table ² for the hardware. The reason to do this would be to reduce the amount of information necessary to output 1 sample. For instance the four polarization states could adequately be represented in 2 bits. Each combination of those 2 bits would correspond to a specific voltage output. Now, what previously took 16 bits to accomplish is done with 2. This would enable the current iteration of the streaming software to drive the WCPS at rates approaching and potentially exceeding a GHz.

²A jump table is a series of instructions conditioned on some value. Essentially, an optimized switch statement

Chapter 4

Conclusion and Outlook

This thesis had two main focuses driven to accommodate two facets of the QEYSSat mission. The first, described in [chapter 2](#), deals with the design of a time of arrival QRNG. While the original scheme has fairly good min entropy as shown in [Figure 2.10](#), it nonetheless could not produce certifiable random bits. This led to implementing an extractor in order to produce 'good' random bits. As there is no physical implementation, the rate of this generator is below the QEYSSat requirement and instead the true random bits are used to seed a fast CSPRNG as an intermediary solution for the QEYSSat ground station. The post processing scheme presented is flexible and can be used with a system having 1 to n SPADs. Additionally, real time corrections to the characteristic curve at the heart of the algorithm can occur as desired. One possible way to do this would be to monitor the min entropy of the QRNG, and upon hitting a threshold recreate/update the characteristic curves. Overall, this algorithm is simple needing only time tags and the resolution of the tagging equipment to operate properly.

Possible improvements can be found by differentiating real counts from noise, whether that noise be from dark counts, after pulses, or cross talk (in the case of an array of detectors). By characterizing the influence of these device imperfections on the resultant measured distribution they could be subsequently mitigated. This could be done by either filtering these malignant counts or by instead looking to minimize the distance between the measured and theoretical distributions. Additionally, from the perspective of purely increasing a potential devices throughput, a hybrid algorithm implementing both a time of arrival and spot of arrival algorithm could be used. It would also be interesting to see if these two degrees of freedom (spatial and temporal) could be exploited in parallel without introducing unwanted correlations in the output stream of random bits.

With a source of random bits in hand, the emphasis shifts to instead producing the random states necessary. This is inherently dependent on the type of source used, and for the WCPS used in the QEYSSat mission, an AWG was necessary. The AWG controlled the electrical signals that would drive the production of these random states in the correct order. Due to the volume of non repeating states being transmitted in a typical run of the protocol (one satellite pass), the ability of the AWG to accept live data being streamed to it was necessary. This also entailed being able to stream the data to the AWG fast enough, which was the subject of [chapter 3](#). Originally, the problem was viewed as a producer-consumer archetype, and this was shown to be inefficient in [Figure 3.7](#). From there the streaming software implementation archetype was reevaluated and instead came to be seen as purely an IO problem. This led to refactoring the software to include more sophisticated IO routines which were more than sufficient to meet the rate required by QEYSSat. Future improvements would be centered on the involved hardware, or decreasing the amount of information that needs to be streamed. Both of these would involve working with the manufacturer on the streaming capabilities of the AWG.

References

- [1] Alvin Ashcraft. File caching, Jan 2021.
- [2] Alvin Ashcraft. Synchronous and asynchronous i/o - win32 apps, Mar 2021.
- [3] Alvin Ashcraft. File buffering - win32 apps, Jun 2022.
- [4] OpenSSL Project Authors. Openssl crypto library, Jan 2023.
- [5] Lawrence Bassham, Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Stefan Leigh, M Levenson, M Vangel, Nathanael Heckert, and D Banks. A statistical test suite for random and pseudorandom number generators for cryptographic applications, 2010-09-16 2010.
- [6] Charles Bennett, Gilles Brassard, Claude Crépeau, and Ueli Maurer. Generalized privacy amplification. *Information Theory, IEEE Transactions on*, 41:1915 – 1923, 12 1995.
- [7] Charles H. Bennett and Gilles Brassard. Quantum cryptography: Public key distribution and coin tossing. *Theoretical Computer Science*, 560:7–11, December 2014.
- [8] J-P Bourgoin, E Meyer-Scott, B L Higgins, B Helou, C Erven, H Hübel, B Kumar, D Hudson, I D’Souza, R Girard, R Laflamme, and T Jennewein. A comprehensive design and performance analysis of low earth orbit satellite quantum communication. *New Journal of Physics*, 15(2):023006, feb 2013.
- [9] Bevin Brett. Memory performance in a nutshell, 2016.
- [10] Karl Bridge, Michael Satran, Thomas Jones, Kent Sharkey, David Coulter, and Drew batchelor. Scheduling Priorities, Jan 2021.

- [11] Cédric Bruynsteen, Tobias Gehring, Cosmo Lupo, Johan Bauwelinck, and Xin Yin. 100-gbit/s integrated quantum random number generator based on vacuum fluctuations. *PRX Quantum*, 4:010330, Mar 2023.
- [12] Simon Carrier, Michel Labrecque-Dias, Ramy Tannous, Pascal Gendron, Frédéric Nole, Nicolas Roy, Tommy Rossignol, Frédéric Vachon, Samuel Parent, Thomas Jennewein, Serge Charlebois, and Jean-François Pratte. Towards a multi-pixel photon-to-digital converter for time-bin quantum key distribution. *Sensors*, 23(7), 2023.
- [13] J.Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.
- [14] Raymond Chen. How do `file_flag_sequential_scan` and `file_flag_random_access` affect how the operating system treats my file?, Jan 2012.
- [15] John C.B Cooper. The poisson and exponential distributions, 2005.
- [16] Ted Dunning and Otmar Ertl. Computing extremely accurate quantiles using t-digests, 2019.
- [17] Latest editor: T. Canens. Cpp reference, Sept 2023.
- [18] Michael J. Evans and Jeffrey S. Rosenthal. *Probability and Statistics - The Science of Uncertainty, Second Edition*. W. H. Freeman, 2023.
- [19] E.W.Dijkstra. Cooperating sequential processes (ewd123 lecture notes), 1965.
- [20] M. Frigo and S.G. Johnson. The design and implementation of fftw3. *Proceedings of the IEEE*, 93(2):216–231, 2005.
- [21] Dr. Paul Godin, Dr. Brendon L. Higgins, Surgeon (Paul) Oh, Ramy Tannous, and Wilson Wu. Qogs quantum subsystem detailed design document. Technical report, University of Waterloo IQC, 2022.
- [22] Robert M. Gray. Toeplitz and circulant matrices: A review. *Foundations and Trends® in Communications and Information Theory*, 2(3):155–239, 2006.
- [23] Robert H. Hadfield. Single-photon detectors for optical quantum information applications. *Nature Photonics*, 3(12):696–705, December 2009.
- [24] W. Heisenberg. Über den anschaulichen Inhalt der quantentheoretischen Kinematik und Mechanik. *Zeitschrift für Physik*, 33:879–893, 1925.

- [25] Miguel Herrero-Collantes and Juan Carlos Garcia-Escartin. Quantum random number generators. *Rev. Mod. Phys.*, 89:015004, Feb 2017.
- [26] Ortwin Hess and Tilmann Kuhn. Spatio-temporal dynamics of semiconductor lasers: Theory, modelling and analysis. *Progress in Quantum Electronics*, 20(2):85–179, 1996.
- [27] Won-Young Hwang. Quantum key distribution with high loss: Toward global secure communication. *Phys. Rev. Lett.*, 91:057901, Aug 2003.
- [28] R. Impagliazzo, L. A. Levin, and M. Luby. Pseudo-random generation from one-way functions. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC '89, page 12–24, New York, NY, USA, 1989. Association for Computing Machinery.
- [29] T. Jennewein, J. P. Bourgoin, B. Higgins, C. Holloway, E. Meyer-Scott, C. Erven, B. Heim, Z. Yan, H. Hübel, G. Weihs, E. Choi, I. D'Souza, D. Hudson, and R. Laflamme. QEYSSAT: a mission proposal for a quantum receiver in space. In Zameer U. Hasan, Philip R. Hemmer, Hwang Lee, and Charles M. Santori, editors, *Advances in Photonics of Quantum Computing, Memory, and Communication VII*, volume 8997, page 89970A. International Society for Optics and Photonics, SPIE, 2014.
- [30] Thomas Jennewein, Ulrich Achleitner, Gregor Weihs, Harald Weinfurter, and Anton Zeilinger. A fast and compact quantum random number generator. *Review of Scientific Instruments*, 71:1675–1680, 04 2000.
- [31] Richard Jiang. Race conditions; deadlocks (cse332 lecture notes), 2020.
- [32] Pouyan Keshavarzian, Karthick Ramu, Duy Tang, Carlos Weill, Francesco Gramuglia, Shyue Seng Tan, Michelle Tng, Louis Lim, Elgin Quek, Denis Mandich, Mario Stipčević, and Edoardo Charbon. A 3.3 gbps spad-based quantum random number generator, 2022.
- [33] Eden Kim. An introduction to solid state drive performance, evaluation and test. Technical report, Solid State Storage Initiative, 2013.
- [34] Donald E. Knuth. *The Art of Computer Programming*. Addison Wesley Longman, 3 edition, 1997.
- [35] Pieter Kok and Brendon W. Lovett. *Introduction to Optical Quantum Information Processing*. Cambridge University Press, 2010.

- [36] Day-Uei Li, Jochen Arlt, Justin Richardson, Richard Walker, Alex Buts, David Stoppa, Edoardo Charbon, and Robert Henderson. Real-time fluorescence lifetime imaging system with a 32×32 $0.13\mu\text{m}$ cmos low dark-count single-photon avalanche diode array. *Opt. Express*, 18(10):10257–10269, May 2010.
- [37] R. Loudon. *The Quantum Theory of Light*. OUP Oxford, 2000.
- [38] Xiongfeng Ma, Feihu Xu, He Xu, Xiaoqing Tan, Bing Qi, and Hoi-Kwong Lo. Postprocessing for quantum random-number generators: Entropy evaluation and randomness extraction. *Phys. Rev. A*, 87:062327, Jun 2013.
- [39] Gleb M. Akselrod Michael A. Wayne, Evan R. Jeffrey and Paul G. Kwiat. Photon arrival time quantum random number generation. *Journal of Modern Optics*, 56(4):516–522, 2009.
- [40] Microsoft. Diskspd, Sept 2021.
- [41] Microsoft. GetLastError function (errhandlingapi.h), June 2021.
- [42] Microsoft. GetOverlappedResult function (ioapiset.h), Sept 2022.
- [43] Microsoft. Overlapped structure (minwinbase.h), Sept 2022.
- [44] Microsoft. ReadFile function (fileapi.h), Sept 2022.
- [45] Microsoft. CreateFileA function, Feb 2023.
- [46] Noriyuki Miyazaki. Crystaldiskinfo, Nov 2023.
- [47] Iyswarya Narayanan, Di Wang, Myeongjae Jeon, Bikash Sharma, Laura Caulfield, Anand Sivasubramaniam, Ben Cutler, Jie Liu, Badriddine Khessib, and Kushagra Vaid. Ssd failures in datacenters: What? when? and why? In *Proceedings of the 9th ACM International on Systems and Storage Conference, SYSTOR '16*, New York, NY, USA, 2016. Association for Computing Machinery.
- [48] Jonathan Netzer. *Proteus Series Arbitrary Waveform Generator/Transceiver Module Platform User Manual Rev. 1.2*. Tabor Electronics.
- [49] Jonathan Netzer. *Proteus Series Arbitrary Waveform Transceiver Programming Manual Rev. 1.4*. Tabor Electronics.

- [50] Rolf Neugebauer, Gianni Antichi, José Fernando Zazo, Yury Audzevich, Sergio López-Buedo, and Andrew W. Moore. Understanding pcie performance for end host networking. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 327–341, New York, NY, USA, 2018. Association for Computing Machinery.
- [51] You-Qi Nie, Hong-Fei Zhang, Zhen Zhang, Jian Wang, Xiongfeng Ma, Jun Zhang, and Jian-Wei Pan. Practical and fast quantum random number generation based on photon arrival time relative to external reference. *Applied Physics Letters*, 104(5):051110, 02 2014.
- [52] Landon Curt Noll, Robert G. Mende, and Sanjeev Sisodiya. Method for seeding a pseudo-random number generator with a cryptographic hash of a digitization of a chaotic system, US Patent 5,732,138, Mar. 1998.
- [53] Victor Y. Pan. *Structured matrices and polynomials: unified superfast algorithms*. Springer-Verlag, Berlin, Heidelberg, 2001.
- [54] Tarikul Islam Papon and Manos Athanassoulis. A parametric i/o model for modern storage devices. In *Proceedings of the 17th International Workshop on Data Management on New Hardware*, DAMON '21, New York, NY, USA, 2021. Association for Computing Machinery.
- [55] Michel Planat, Haret C. Rosu, and Serge Perrine. A survey of finite algebraic geometrical structures underlying mutually unbiased quantum measurements. *Foundations of Physics*, 36(11):1662–1680, November 2006.
- [56] Christopher J Pugh, Sarah Kaiser, Jean-Philippe Bourgoin, Jeongwan Jin, Nigar Sultana, Sascha Agne, Elena Anisimova, Vadim Makarov, Eric Choi, Brendon L Higgins, and Thomas Jennewein. Airborne demonstration of a quantum key distribution receiver payload. *Quantum Science and Technology*, 2(2):024009, June 2017.
- [57] Leo Reyzin. Extractors and the leftover hash lemma, 2011.
- [58] J. J. Sakurai and Jim Napolitano. *Modern Quantum Mechanics*. Cambridge University Press, 3 edition, 2020.
- [59] Nicolas Sangouard, Christoph Simon, Hugues de Riedmatten, and Nicolas Gisin. Quantum repeaters based on atomic ensembles and linear optics. *Rev. Mod. Phys.*, 83:33–80, Mar 2011.

- [60] Elham Sarbazi, Majid Safari, and Harald Haas. Photon detection characteristics and error performance of spad array optical receivers. In *2015 4th International Workshop on Optical Wireless Communications (IWOW)*, pages 132–136, 2015.
- [61] Rebecka Sax, Alberto Boaron, Gianluca Boso, Simone Atzeni, Andrea Crespi, Fadri Grünenfelder, Davide Rusca, Aws Al-Saadi, Danilo Bronzi, Sebastian Kupijai, Hanjo Rhee, Roberto Osellame, and Hugo Zbinden. High-speed integrated qkd system. *Photon. Res.*, 11(6):1007–1014, Jun 2023.
- [62] Mario Stipčević and Çetin Kaya Koç. *True Random Number Generators*, pages 275–315. Springer International Publishing, Cham, 2014.
- [63] Mario Stipčević. Physical defintion of randomness, 2023.
- [64] Mario Stipčević. Physical defintion of randomness, 2023.
- [65] Ivo Straka, Jan Grygar, Josef Hloušek, and Miroslav Ježek. Counting statistics of actively quenched spads under continuous illumination. *J. Lightwave Technol.*, 38(17):4765–4771, Sep 2020.
- [66] Ramy Tannous. *Polarization Entangled Photon Sources for Free-Space Quantum Key Distribution*. PhD thesis, University of Waterloo, 2018.
- [67] Tektronics. Xyzs of signal generators, 2023.
- [68] G. S. Vernam. Cipher printing telegraph systems for secret wire and radio telegraphic communications. *Transactions of the American Institute of Electrical Engineers*, XLV:295–301, 1926.
- [69] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.
- [70] Michael A. Wayne and Paul G. Kwiat. Low-bias high-speed quantum random number generator via shaped optical pulses. *Opt. Express*, 18(9):9351–9357, Apr 2010.

- [71] W.K. Wootters and W.H. Zurek. A single quantum cannot be cloned. *Nature*, 299(5886):802–803, 1982.
- [72] William K Wootters and Brian D Fields. Optimal state-determination by mutually unbiased measurements. *Annals of Physics*, 191(2):363–381, 1989.
- [73] Wilson Wu. Weak coherent pulse source and polarization control for quantum key distribution, 2023.
- [74] Qiurong Yan, Baosheng Zhao, Qinghong Liao, and Nanrun Zhou. Multi-bit quantum random number generation by measuring positions of arrival photons. *Review of Scientific Instruments*, 85(10):103116, 10 2014.

APPENDICES

.1 SPAD Camera

To test the QRNG, data was collected from a SPAD camera containing an 8x8 array of SPAD's from Micro Photon Devices (MPD). Only information pertaining to the Photon Detection efficiency and Dark Count Rate was made available.

SINGLE-PHOTON SPAD ARRAY - SPA Test report



Date: May 18, 2019

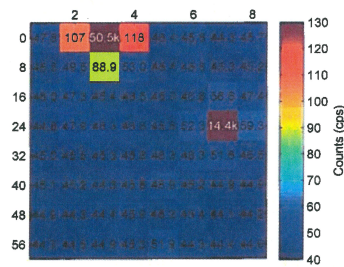
Product Identification

Product Type: \$PDA-8X8UV-DO.
Product Description: 2D SPAD Array - 8x8 pixels - Direct Outputs.
Serial Number: 00003.

Performance Data

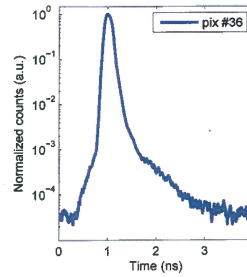
Dark Count Rate

Integration time: 300 s
Detector temperature during the test: 27.3 °C.



Temporal Response

FWHM across the pixels: 162.5 ps $\pm$ 10.6 ps.
Skew between pixels: < 750 ps



Hold-off

Binary code	00	01	10	11
Measured value (ns)	42	63	114	255

Figure 1: While most pixels have roughly a hundred dark counts, pixel 3 and 31 have substantially higher DCR of several thousand kHz.

.2 Characteristic Curve

Pixel 31 characteristic curve values, and bins, detailed below to clarify the adjustable aspect of the QRNG bins. The characterization curve does not explicitly store the values of 0 and ∞ , as any delta that is not greater than the first bin edge belongs to the zeroth bin. Likewise, a delta that is not less than any bin edge belongs to the last bin.

Table 1: Characterization Curve Values for Pixel 31

Bin Edge	Bin	Bin Width
1.422 μs	(0, 1.422 μs)	1.422 μs
2.785 μs	(1.422 μs , 2.785 μs)	1.363 μs
4.138 μs	(2.785 μs , 4.138 μs)	1.353 μs
5.571 μs	(4.138 μs , 5.571 μs)	1.432 μs
6.979 μs	(5.571 μs , 6.979 μs)	1.409 μs
8.408 μs	(6.979 μs , 8.408 μs)	1.428 μs
9.824 μs	(8.408 μs , 9.824 μs)	1.417 μs
11.339 μs	(9.824 μs , 11.339 μs)	1.515 μs
12.840 μs	(11.339 μs , 12.840 μs)	1.501 μs
14.301 μs	(12.840 μs , 14.301 μs)	1.460 μs
15.849 μs	(14.301 μs , 15.849 μs)	1.549 μs
17.305 μs	(15.849 μs , 17.305 μs)	1.455 μs
18.893 μs	(17.305 μs , 18.893 μs)	1.588 μs
20.542 μs	(18.893 μs , 20.542 μs)	1.649 μs
22.161 μs	(20.542 μs , 22.161 μs)	1.619 μs
23.765 μs	(22.161 μs , 23.765 μs)	1.604 μs
25.416 μs	(23.765 μs , 25.416 μs)	1.651 μs
27.094 μs	(25.416 μs , 27.094 μs)	1.678 μs
28.745 μs	(27.094 μs , 28.745 μs)	1.652 μs
30.477 μs	(28.745 μs , 30.477 μs)	1.730 μs
32.222 μs	(30.477 μs , 32.222 μs)	1.745 μs
33.989 μs	(32.222 μs , 33.989 μs)	1.767 μs
35.783 μs	(33.989 μs , 35.783 μs)	1.793 μs
37.689 μs	(35.783 μs , 37.689 μs)	1.906 μs
39.561 μs	(37.689 μs , 39.561 μs)	1.872 μs
41.565 μs	(39.461 μs , 41.565 μs)	2.004 μs

Bin Edge	Bin	Bin Width
43.601 μs	(41.565 μs , 43.601 μs)	2.036 μs
45.711 μs	(43.601 μs , 45.711 μs)	2.110 μs
47.832 μs	(45.711 μs , 47.832 μs)	2.122 μs
50.083 μs	(47.832 μs , 50.083 μs)	2.250 μs
52.378 μs	(50.083 μs , 52.378 μs)	2.295 μs
54.694 μs	(52.378 μs , 54.694 μs)	2.316 μs
56.935 μs	(54.694 μs , 56.935 μs)	2.219 μs
59.348 μs	(56.935 μs , 59.348 μs)	2.434 μs
61.761 μs	(59.438 μs , 61.761 μs)	2.413 μs
64.575 μs	(61.761 μs , 64.575 μs)	2.813 μs
67.203 μs	(64.575 μs , 67.203 μs)	2.628 μs
69.914 μs	(67.203 μs , 69.914 μs)	2.711 μs
72.974 μs	(69.914 μs , 72.974 μs)	3.060 μs
76.109 μs	(72.974 μs , 76.109 μs)	3.135 μs
79.377 μs	(76.109 μs , 79.377 μs)	3.268 μs
82.819 μs	(79.377 μs , 82.819 μs)	3.441 μs
86.435 μs	(82.819 μs , 86.435 μs)	3.616 μs
90.128 μs	(86.435 μs , 90.128 μs)	3.693 μs
93.910 μs	(90.128 μs , 93.910 μs)	3.782 μs
97.967 μs	(93.910 μs , 97.967 μs)	4.057 μs
102.409 μs	(97.967 μs , 102.409 μs)	4.442 μs
106.984 μs	(102.409 μs , 106.984 μs)	4.575 μs
111.821 μs	(106.984 μs , 111.821 μs)	4.834 μs
117.166 μs	(111.821 μs , 117.166 μs)	5.345 μs
122.823 μs	(117.166 μs , 122.823 μs)	5.657 μs
128.775 μs	(122.823 μs , 128.775 μs)	5.952 μs
135.283 μs	(128.775 μs , 135.283 μs)	6.509 μs
142.482 μs	(135.283 μs , 142.482 μs)	7.199 μs
150.147 μs	(142.482 μs , 150.147 μs)	7.665 μs
159.062 μs	(150.147 μs , 159.062 μs)	8.915 μs
169.542 μs	(159.062 μs , 169.542 μs)	10.480 μs
181.030 μs	(169.542 μs , 181.030 μs)	11.489 μs
194.732 μs	(181.030 μs , 194.732 μs)	13.702 μs
211.962 μs	(194.732 μs , 211.962 μs)	17.230 μs
233.836 μs	(211.962 μs , 233.836 μs)	21.873 μs

Bin Edge	Bin	Bin Width
264.352 μs	(233.836 μs , 264.352 μs)	30.516 μs
317.619 μs	(264.352 μs , 317.619 μs)	53.267 μs
∞	(317.619 μs , ∞)	∞