1. Quick review of the implementation (System.py)

   Full code can be found at https://github.com/KwiatQIM/quantum-key-distri bution
   To download: **git clone https://github.com/KwiatQIM/quantum-key-distribution.git**

   A bit cleaner code is at https://github.com/KwiatQIM/full-qkd-system
   To download: **git clone https://github.com/KwiatQIM/full-qkd-system.git**

   If running on lab server use virtual environment and get requirements from /storage/Laurynas/quantum-key-distribution/requirements.txt

   To activate existing virtual environment go to:

   /storage/Laurynas/quantum-key-distribution

   And type:

   source qkd/bin/activate

   Two threads for Alice and Bob are created as shown in Figure 1 by Main thread which also sets file directories and other parameters. The main thread is blocked and is waiting for both threads to finish loading timetag and polarization values, loading and applying delays and finally sorting data. As Alice/Bob finished they are blocked and will be waiting for main thread which now calculates and distributes optimal frame size. (The optimal frame size is hardcoded but can be calculated by uncommenting optimal frame size section in System.py). Furthermore, Alice/Bob are released and calculates frame occupancies and locations that is simply a string of positions of 1s in the frames. Moreover, the work in Alice/Bob threads are done so they are killed and now main thread makes frame occupancy and location strings of equal size, finds mutual frames with single occupancies and calculates mutual bases string and extracts actual measured polarizations. By comparing the latter, one is able to estimate QBER which uses strong assumption that all non-matching polarizations in mutual bases string are errors introduced by Eve. In order to get rid of this corrupted data the non-matching bases are thrown out.

   Moreover, for the error correction we must have channel statistics which would allow us to use belief propagation and correct all errors hence system announces some fraction of original string to obtain statistics. Next, the LDPC encoding procedure applied and syndrome values (parity check values) are transferred to Bob from Alice along with the parity check matrix.
   Bob using this information is able to decode the string and eventually both parties should have same keys. These keys are made of two parts – corrected mutual bases polarization string and corrected locations of ones in mutual frames. As Eve still knows information about the keys the privacy amplification is applied in order to shuffle keys and minimize Eve's knowledge.

2. Getting right data format

In order for the system to work, the timetager data must have 2 columns. The first column must contain channel number (polarization) and the second column represents actual timetag value in bins.

NOTE: both columns contain data from both Alice and Bob parties.

Q: If the timetag value is in seconds?

A: The timetag value must be converted in a bin number by dividing value in seconds by the length of the laser pulse on the crystal as that is synchronized laser pulse which indicates that both Alice and Bob start new period of time which ideally will contain Alice and Bob photon from entangled pair and therefore the information (entropy) can be extracted.

3. Delays

Both Alice and Bob are independent parties, hence they have different wiring and therefore there is some constant offset between Alice and Bob channels which can be found running Delays.py. The delays are saved to ./resultsLaurynas/Delays/delays.npy this can be found and adjusted at the very end of Delays.py file.

Parameters:

Results: The saved Numpy array represents offsets in bins between two same polarization channels.

4. Setting the system parameters

Once the delays for the dataset are found and data is in valid format one can adjust the system parameters, which are defined in the grey box saying “PARAMETERS”. Let me describe every single of them.

a. raw_ file_dir - it is the directory of the raw text file which has data in the format described in Section 2.
b. alice_channels - defines Alice polarizations in the following order [H,V,A,D].
c. bob_channels - same as above just for Bob.
d. Resolution - the resolution of the bins (i.e. the size of the bin in seconds). By default, as described above we use 260 picoseconds which is the size of a laser pulse on the crystal and is determined by the laser frequency.
e. coincidence_window_radius – was previously used in the announcing scheme which would correct detector jitter in the timetags (not in use anymore).
f. announce_fraction – the fraction of the original string to be announced
g. announce_binary_fraction – the fraction of the mutual bases polarization (binary) string to be announced
h. data_factor – the fraction of the actual dataset to be taken.
i. Optimal_frame_size – the optimal frame size which contains the most coincidence counts and the least garbage coming from uncorrelated events. Also it would contain the most entropy. (For this particular dataset I found it to be 8)
j. column_weight – this is used in the LDPC parity check matrix where it defines how many parities use the same bit value.
k. row_weight – this is used in the LDPC parity check matric where it defines how many bit values does a parity check have.
l. fraction_parity_checks – is the fraction of original string which determines the number of parity checks.

NOTE: some of the parameters are leftovers from previous implementations. Also, sometimes the functions do not take the specified parameters but has them defined in the signature by default (be careful with that!).

If you have already saved your processed data to .npy format (as it works way faster than reprocessing raw .txt file every single time), you can just load using load_data method it should already contain some directories to .npy files. The original .txt file is separated to timetag and channel .npy files which contain information about both Alice and Bob data. P.S in order to save .npy just load text file using loadtxt() and to save use save(), the procedure can be found in load_save_raw_file() method.

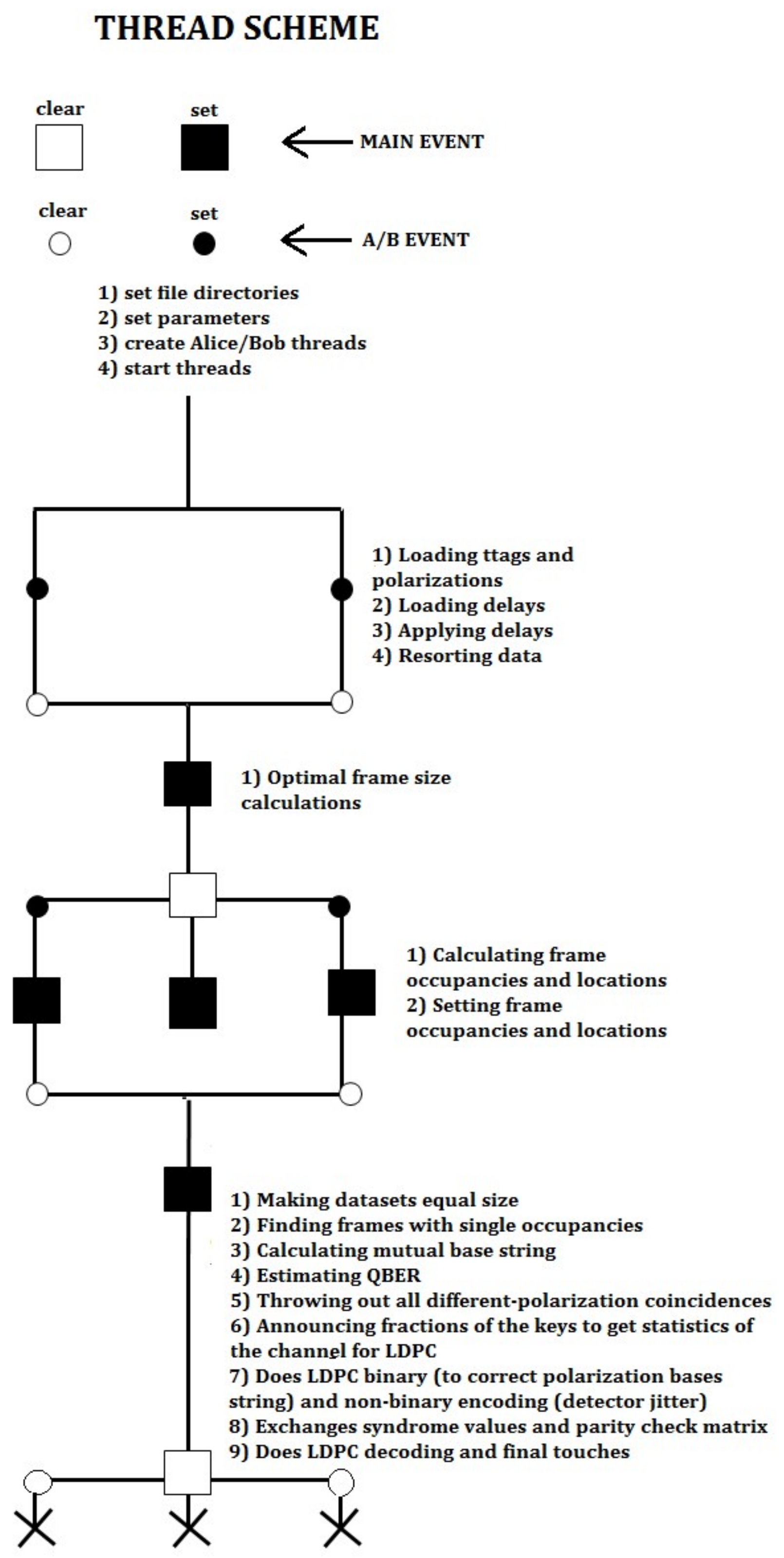
THREAD SCHEME
clear
set
MAIN EVENT
clear
set
A/B EVENT
1) set file directories
2) set parameters
3) create Alice/Bob threads
4) start threads
1) Loading ttags and polarizations
2) Loading delays
3) Applying delays
4) Resorting data
1) Optimal frame size calculations
1) Calculating frame occupancies and locations
2) Setting frame occupancies and locations
1) Making datasets equal size
2) Finding frames with single occupancies
3) Calculating mutual base string
4) Estimating QBER
5) Throwing out all different-polarization coincidences
6) Announcing fractions of the keys to get statistics of the channel for LDPC
7) Does LDPC binary (to correct polarization bases string) and non-binary encoding (detector jitter)
8) Exchanges syndrome values and parity check matrix
9) Does LDPC decoding and final touches

Figure 1

Email me if you have questions: laurynas.tamulevicius@gmail.com

5. LDPC correcting

   Main algorithm is in SlepianWolf.py and some prerequisite procedures are in SW_prep.py.
   In order to start LDPC, first you must encode Alice string. You can do it following these steps:

   1. ENCODING: Create parity check matrix (I recommend using randomMatrix() method from SW_prep.py as that was the best way to find good parity check matrices). It will take total string length which is the number of bits you want to encode, number of parity check equations or simply number of parity checks which is the number of rows in the parity check matrix and also number of syndrome values (which defines how many secret bits you revealed) (there are a lot of stuff which is named in 20 different names but has the same meaning) and also matrix has two other parameters – column weight and row weight. The former can be set and I recommend to make it 3 as that's the most efficient value from my experience but it might be the case that for larger strings this goes up. The row weight is set automatically and in order to figure it out you should go to randomMatrix method. For more info, check the belief propagation slides.
   2. The next step is to encode the original string and obtain syndrome values which are simply checksums of associated bits.
   3. Now you can announce these syndrome values to Bob and he will try to decode it.
   4. DECODING: You first need to find transition matrix which is simply counting the probabilities of Bob getting the error in adjacent bins. This requires piece of Alice's string and as I talked to Daniel his suggestion was to have some interval of time prior QKD which would allow to obtain channel statistics.
   5. Prior_probability_matrix is just the same transition matrix which assigns to all Bob's bits the probabilities of letters from transition matrix. (This is where conditional probability comes from in belief propagation).
   6. Having all these elements we are ready to create belief propagation system which would propagate all the values as described in the slides and luckily approach correct values. The actual algorithm is complex and I have a small activity diagram in my notebook (look for green tag) which might help to follow it. Also here you can select decoder, I was using bp-fft which sometimes diverges and throws bunch of exceptions but I could not follow log-bp-ftt which was working but converting probabilities to log base).
   7. Having everything set up now you can decode Bob's string which will show remaining errors in the string after every iteration.
   8. During the last days I was testing the retained entropy artificially generating errors according to Poisson (in the GenerateError.py) and manual (GenerateErrorManualDistr.py) probability distribution in the adjacent bins.
      You can simply use it setting Alice string directory, setting alphabet size, setting fraction of errors and directory for Bob's string with the errors. Some plots might be found in Docs folder. Also in manual probability distribution you can enter the probabilities of getting +-1 bin offset (I left it at 50/50% this means that if there's 30% error you would get 15% of by +1 and other 15% of by -1 bin).

Email me if you have questions: laurynas.tamulevicius@gmail.com