## NETWORK LAB REPORT

NAME : Dibyendu Mukhopadhyay

CLASS : BCSE-III

ROLL NO. : 001710501077

GROUP : A3

ASSIGNMENT NO. : 2

# PROBLEM STATEMENT:

# Implement three data link layer protocols, Stop and Wait, Go Back N Sliding Window and Selective Repeat Sliding Window for flow control.

# SUBMISSION DUE : 10TH FEBRUARY,2020

# REPORT SUBMITTED : 24TH FEBRUARY,2020

This report has three sections: Section 1 contains Stop and Wait protocol, section 2 contains Go Back n protocol and section 3 contains Selective Repeat Sliding windows protocol.

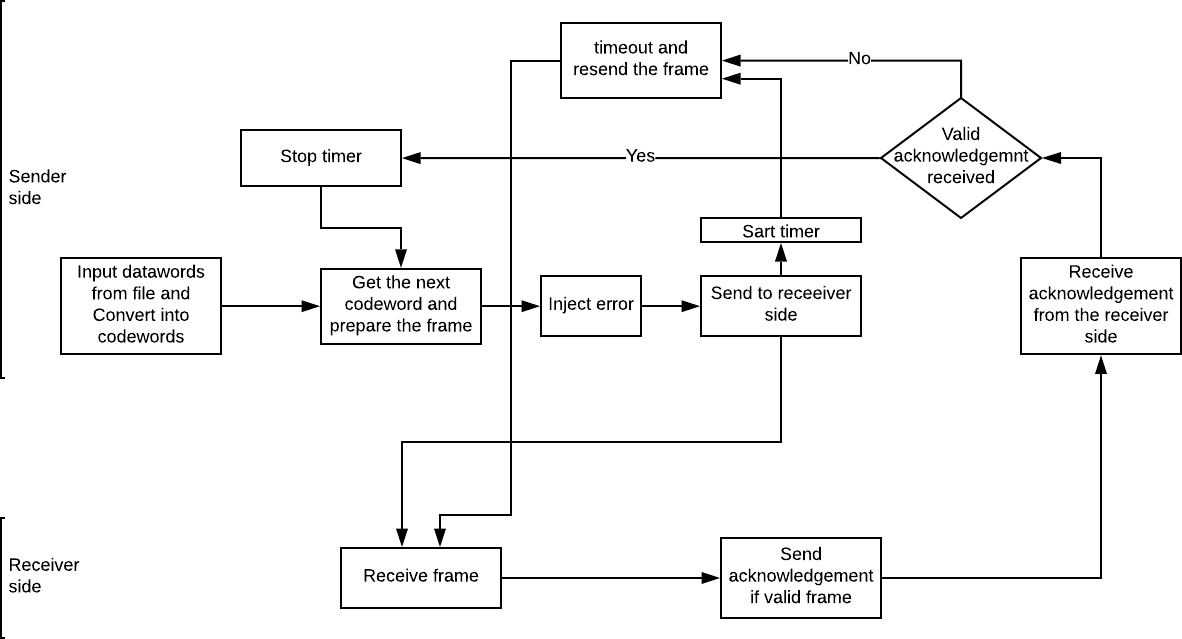
# SECTION 1: Stop and wait arq

## Design

The protocol is designed using two files- snw\_sender.py and snw\_receiver.py.

1. **snw\_sender.py**: This file handles the sender side. It takes data input from file, converts the data into number of codewords of length of given frame size using the VRC algorithm. Then the program sends the frames one by one to the receiver. While sending the frame, it also introduces the error to the frame in a few cases. It also receives the acknowledgements from the receiver side and in case of error it resends the frame.
2. **snw\_receiver.py**: This file handles the receiver side. It accepts the frames sent from the sender side, checks for error (if any), and if it is the expected frame, it sends the acknowledgment to the sender end.

The design of the implementation is shown in the diagram (1)



*Figure 1: design of the implementation of stop and wait arq algorithm*

Some important parameters used in the implementation are described below:

1. **Frame size**: 16 bits
2. **Dataword size**: 8 bits
3. **Frame format**: sequence number of frame followed by the stuffed zero and the codeword at the end. The 8 bit data word is converted into 9 bit codeword using the vrc algorithm. Then 6 0’s are stuffed before it and finally the sequence number (0 or 1) is added before it to generate the 16 bit frame.
4. **Error detection algorithm used**: vrc or simple pairity checker.
5. **Acknowledgement format**: 1 bit acknowledgement specifying the next expected frame.
6. **Input file format**: bit stream of 0’s and 1’s.

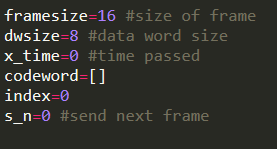
## Implementation

# In this problem statement, the entire program is implemented in Python3.

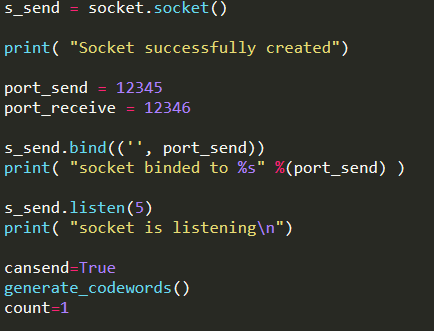
# The detailed method description is written below with a suitable code snippets along with comments for better understanding code overview.

* *snw\_sender.py*

These are the global variables which is declared globally.

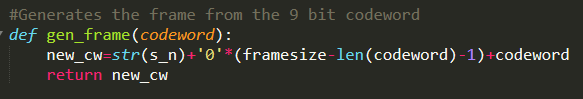
****

Socket programming are implemented globally also which helps to interact between sender and receiver. Here, the codewords are send via port.

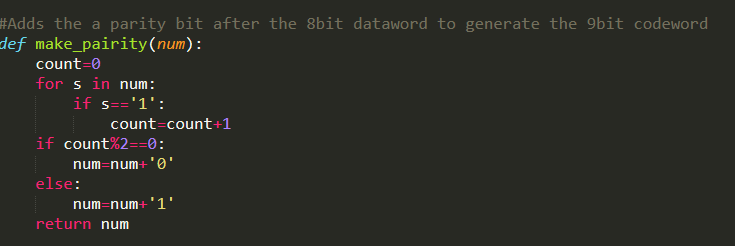
****

**Function Definitions:**

* This method generates the frame from the 9 bit codeword.

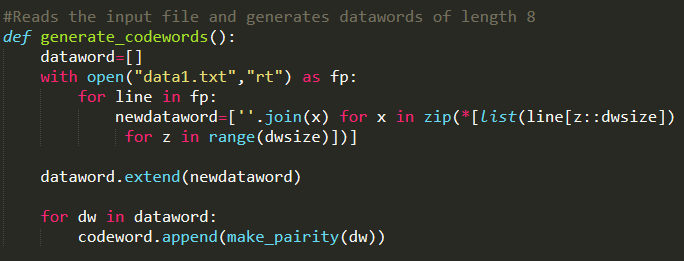


* This method adds the a parity bit after the 8 bit dataword to generate the 9bit codeword

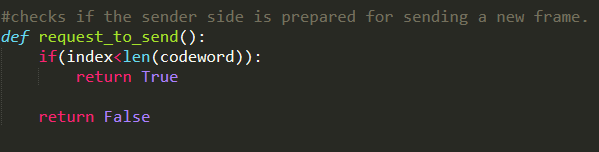


* This function reads the input file and generates datawords of length 8

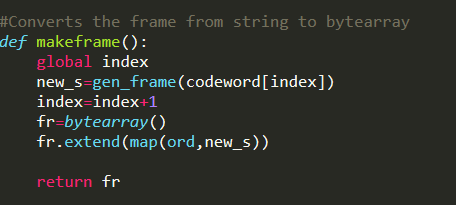
and then return it .



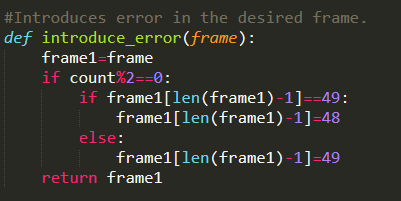
* This module checks if the sender is prepare for sending the new frames.



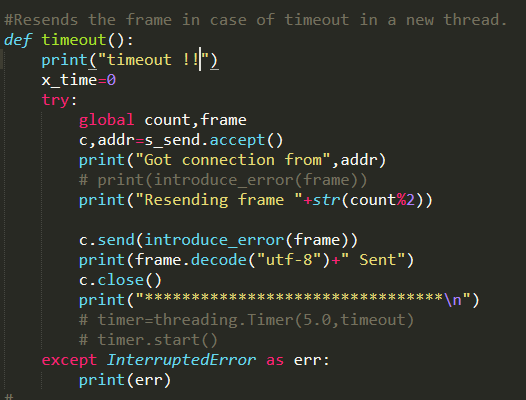
* This method actually converts the frames from string to bytearray.



* This module introduces the error in the desired frame.



* This method helps to resend the frame in case of timeout in a new thread.

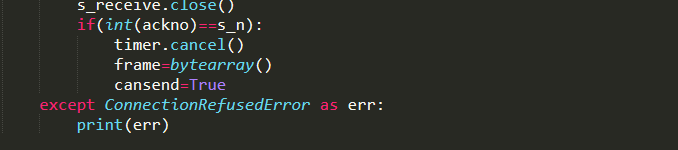


* **Main Thread :**

The **while()** loop runs infinitely to run the send and receive operations whenever frame is ready and acknowledgment is received. When the sender is ready to send and there is a request to send, the sending socket checks for a connection with receiver port. If the connection is found sender creates a new frame from the dataword and introduces error (if any) to it. Then the sender sends the frame to the receiver port using the sending socket and the timer starts and cansend is turned **false**.

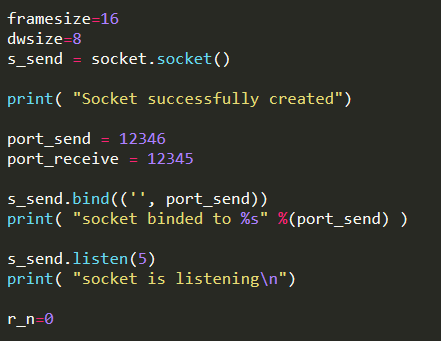
The receiver socket searches for a connection from the sending socket in the receiver side. If there is any connection, the socket accepts the acknowledgment. If the acknowledgement is not corrupted and it is same as the next ready frame to be sent, then the timer is stopped and cansend is turned true.



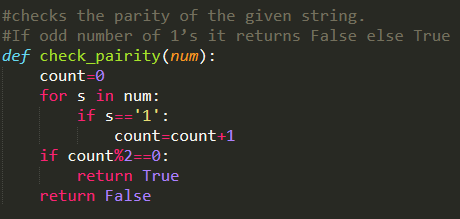


* *snw\_receiver.py*

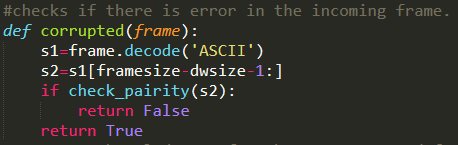
Global variables declaration and socket modules



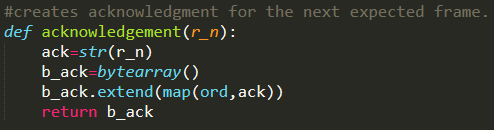
* This method checks the parity of the given string. If the odd number of 1’s, it returns False else True.



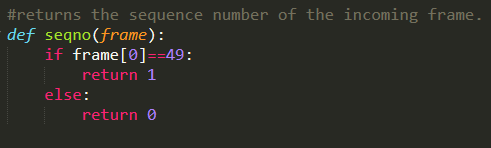
* This method checks whether there is an error in an incoming frame.



* This method creates the ACK for the next expected frame.



* This function defines the sequence number of the incoming frame.

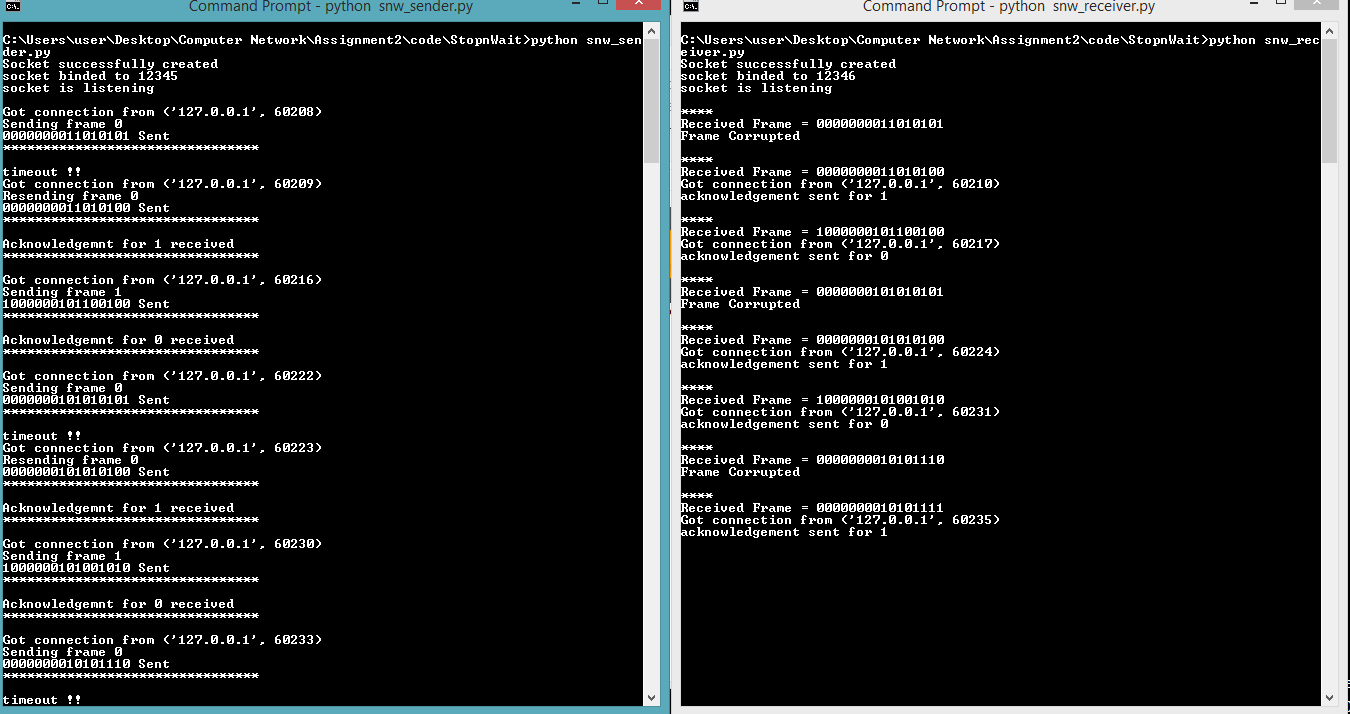


* **Main Thread :**

The receive socket in the receiver side searches for connection from port in the sender side. If a connection is found, it receives the frame, checks if corrupted and if the sequence number is same as expected. Then a sender socket is created that sends the acknowledgement to the sender side.



* **OUTPUT**

****

* + **Results**:

The performance of the above algorithm is measured in terms of throughput. The entire data file consists of 20 datawords. So, for sending 20 frames in average 40 attempts were taken considering insertion of random errors. Considering timeout after 5 seconds average propagation delay is 2.5 seconds.

## Analysis:

The program may have some possible bugs due to the lack of randomness in injecting the error. Again, the program is implemented for one sender and receiver that can be extended up to multiple senders and receivers. The last few bits from the input file is discarded to make the dataword size same. This bug can be overcome by padding of 0’s or 1’s.

## Comment:

The assignment was very helpful in better understanding of the implementation of one of the most popular flow control algorithms in noisy channel. It’s level can be rated as moderately high.

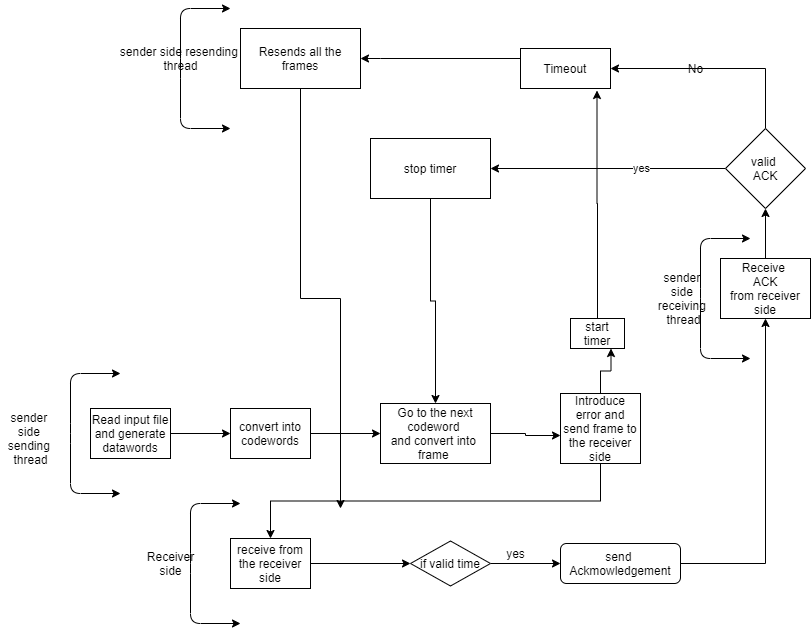
# SECTION 2 : Go back n arq

## Design

The protocol is designed using two files- gbn\_sender.py and gbn\_receiver.py.

1. **gbn\_sender.py**: This file handles the sender side. It takes data input from file, converts the data into number of codewords of length of given frame size using the vrc algorithm. Then the program sends the frames one by one to the receiver. While sending the frame it also introduces error to the frame in a few cases. It also receives the acknowledgements from the receiver side and in case of error it resends the frame.
2. **gbn\_receiver.py**: This file handles the receiver side. It accepts the frames sent from the sender side, checks for error (if any), and if it is the expected frame, it sends the acknowledgment to the sender end.

The design of the implementation is shown in the diagram (2)



*Figure 2: design of the implementation of go back n arq algorithm*

Some important parameters used in the implementation are described below:

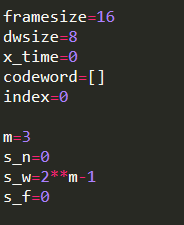
1. **Frame size**: 16 bits
2. **Dataword size**: 8 bits
3. **Frame format**: sequence number of frame following the stuffed zeros and the codeword at the end. The 8 bit data word is converted into 9 bit codeword usong vrc algorithm. Then the sequence number of the frame is added before it and finally zeros are stuffed to make the frame size 16.
4. **Error detection algorithm used**: vrc or simple pairity checker.
5. **Acknowledgement format**: 1 bit acknowledgement specifying the next expected frame.
6. **Input file format**: bit stream of 0’s and 1’s.
7. **Window size**: in this case m=3. Hence window size *𝑠\_𝑤 = 2𝑚 − 1 = 7*

## Implementation

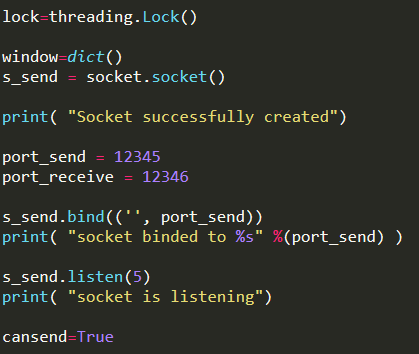
The algorithm has been implemented using python3. The details are given below.

* + *gbn\_sender.py*

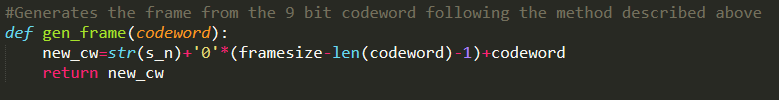
Global variables and modules



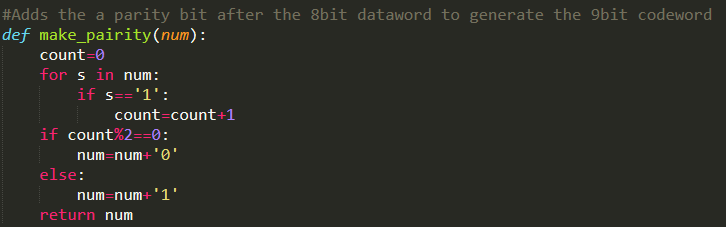
Socket functionality



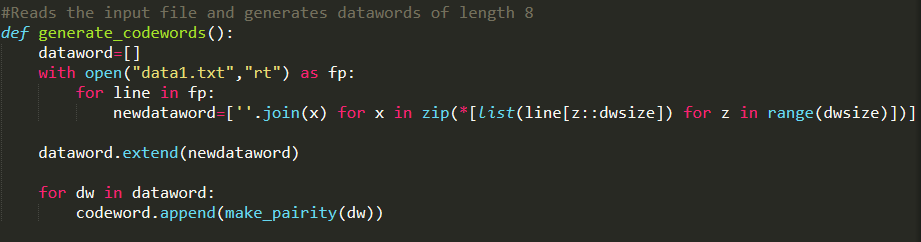
* + This method generates the frame from the 9 bit codeword.



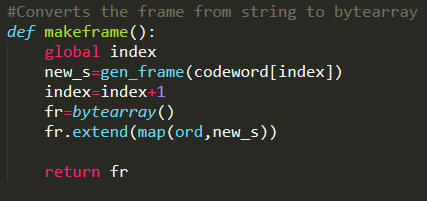
* + This function adds the parity bit after the 8 bit data word to generate 9 bit codeword.



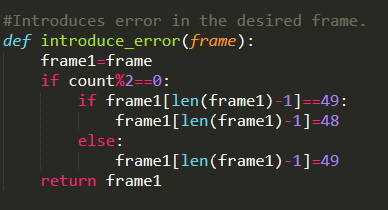
* + This function reads the input data file and generates the dataword of length 8.



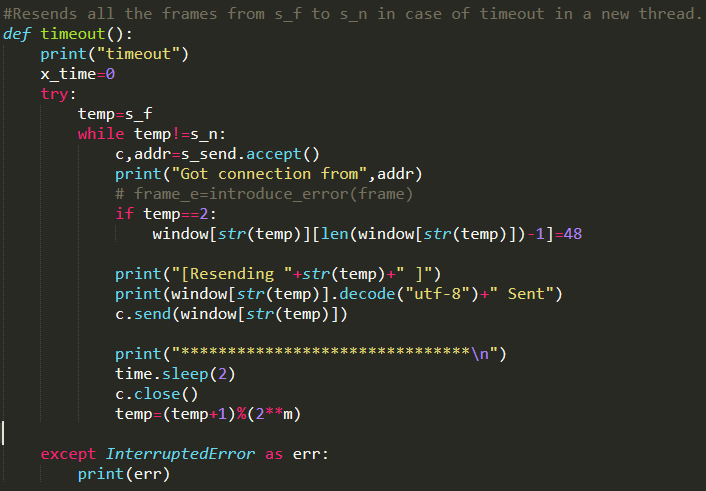
* + It converts the frame from string to bytearray.



* + This module introduces the error in the desired frame.



* + Resends all the frames from s\_f to s\_n in case of timeout in a new thread.



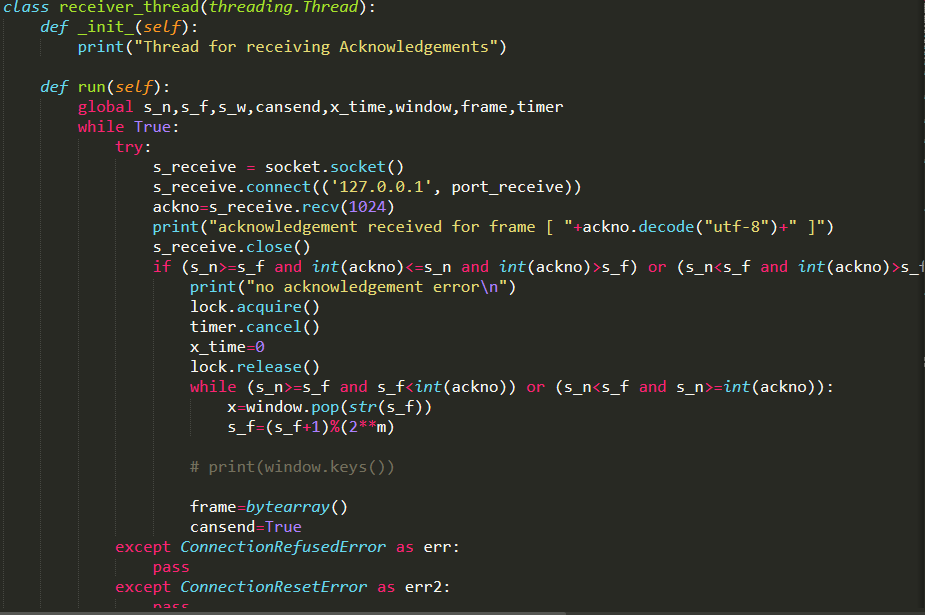
* + **Threading class**
* **class sending\_thread(threading.Thread):**

This class handles the sending of the data frames from the sender side to the receiver side. The **while()** loop runs infinitely to run the send operation whenever frames are ready and acknowledgment is received. When the sender is ready to send and there is a request to send, the sending socket checks for a connection with receiver port. If the connection is found sender creates a new frame from the dataword and introduces error (if any) to it. Then the sender sends the frame to the receiver port using the sending socket and the timer starts and cansend is turned false.

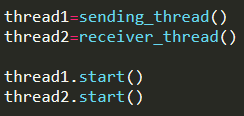


* **class receiver\_thread(threading.Thread)**

This class handles the receiving of the acknowledgement from the receiver side. The receiver socket searches for a connection from the sending socket in the receiver side. If there is any connection, the socket accepts the acknowledgment. If the acknowledgement is not corrupted and it is same or greater than s\_f, then the timer is stopped and cansend is turned true. Again, all the frames from s\_f to the acknowledgement number is purged.

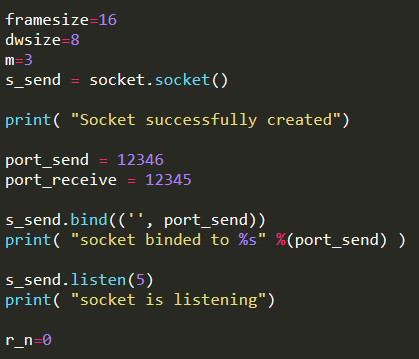
****

* + **Main Thread**

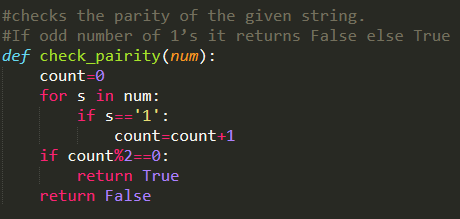
****

* + *gbn\_receiver.py*

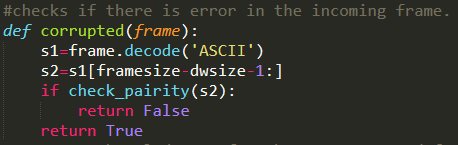
Global variables and modules



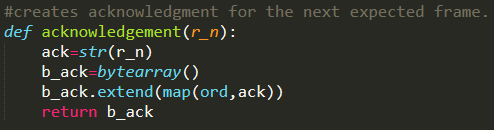
* This method checks the parity of the given string. If the odd number of 1’s, it returns False else True.



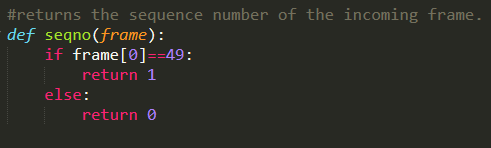
* This method checks whether there is an error in an incoming frame.



* This method creates the ACK for the next expected frame.



* This function defines the sequence number of the incoming frame.

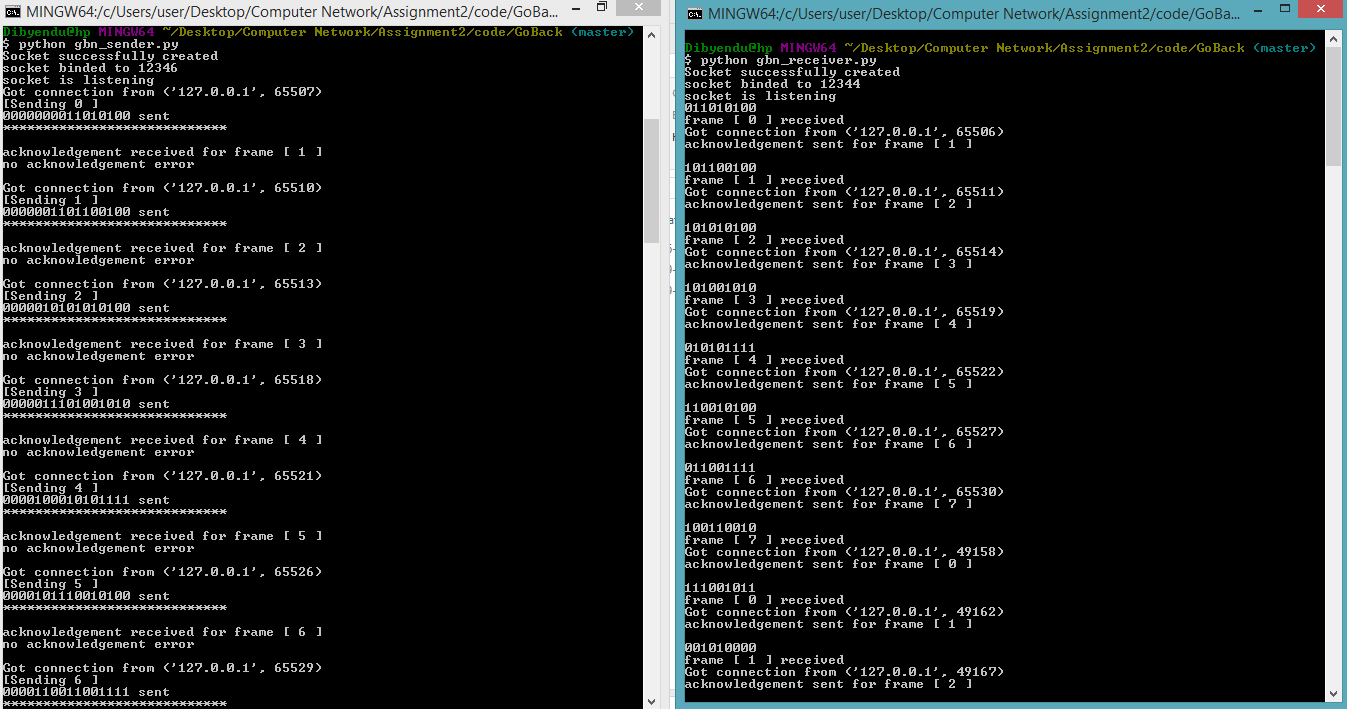


* **Main Thread**

The receive socket in the receiver side searches for connection from a port in the sender side. If a connection is found, it receives the frames, checks if corrupted and if the sequence number is same or more than as expected. Then a sender socket is created that sends the ACK for the next expected frame to the sender side.



* **OUTPUT**

****

* + **Result** :

The performance of the above algorithm is measured in terms of throughput. The entire data file consists of 20 datawords. So for sending 20 frames in average of 65 attempts were taken considering insertion of random errors. Considering timeout after 5 seconds average propagation delay is 2 seconds.

## Analysis:

The program may have some possible bugs due to the lack of randomness in injecting the error. Again, the program is implemented for one sender and receiver that can be extended up to multiple senders and receivers. The last few bits from the input file is discarded to make the dataword size same. This bug can be overcome by padding of 0’s or 1’s.

## Comment:

The assignment was very helpful in better understanding of the implementation of one of the most popular flow control algorithms in noisy channel. Its difficulty level can be rated as extremely high.

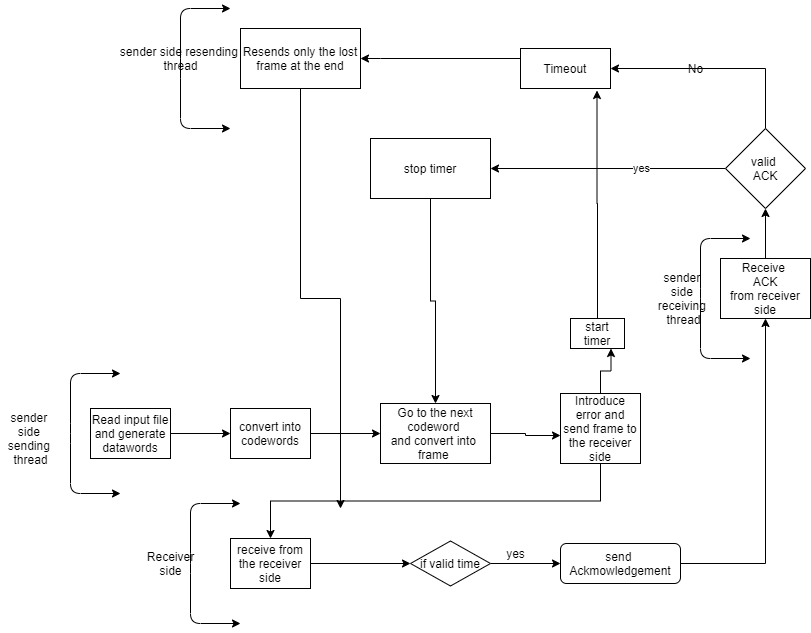
# SECTION 3 : Selective Repeat Sliding Window

# Design

The protocol is designed using two files- gbn\_sender.py and gbn\_receiver.py.

1. **srp\_sender.py**: This file handles the sender side. It takes data input from file, converts the data into number of codewords of length of given frame size using the vrc algorithm. Then the program sends the frames one by one to the receiver. While sending the frame it also introduces error to the frame in a few cases. It also receives the acknowledgements from the receiver side and in case of NAK, it resends the frame at the end after sending only the ACK frame in the sorting way.
2. **srp\_receiver.py**: This file handles the receiver side. It accepts the frames sent from the sender side, checks for error (if any), and if it is the expected frame, it sends the acknowledgment to the sender end.

The design of the implementation is shown in the diagram (3)



*Figure 3: design of the implementation of selective repeat algorithm*

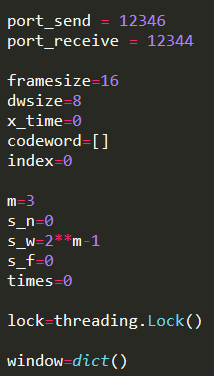
1. **Frame size**: 16 bits
2. **Dataword size**: 8 bits
3. **Frame format**: sequence number of frame following the stuffed zeros and the codeword at the end. The 8 bit data word is converted into 9 bit codeword usong vrc algorithm. Then the sequence number of the frame is added before it and finally zeros are stuffed to make the frame size 16.
4. **Error detection algorithm used**: vrc or simple pairity checker.
5. **Acknowledgement format**: 1 bit acknowledgement specifying the next expected frame.
6. **Input file format**: bit stream of 0’s and 1’s.
7. **Window size**: in this case m=3. Hence window size *𝑠\_𝑤 = 2𝑚 − 1 = 7*

## Implementation

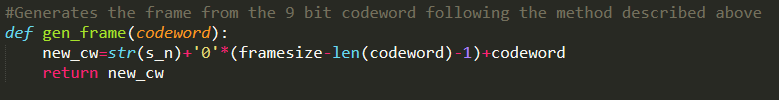
The algorithm has been implemented using python3. The details are given below.

* + *Srp\_sender..py*

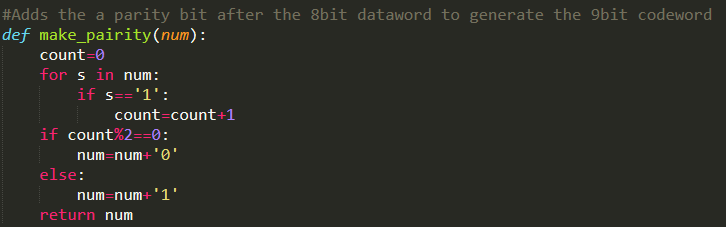
Global variables and modules

****

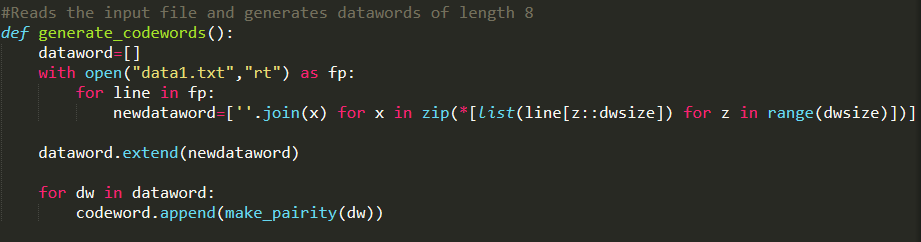
* + This method generates the frame from the 9 bit codeword.



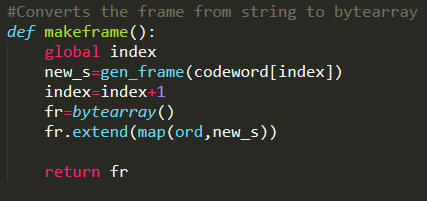
* + This function adds the parity bit after the 8 bit data word to generate 9 bit codeword.



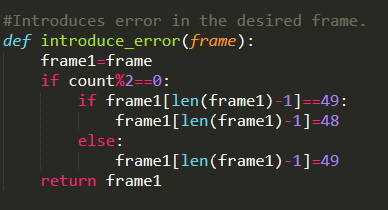
* + This function reads the input data file and generates the dataword of length 8.



* + It converts the frame from string to bytearray.



* + This module introduces the error in the desired frame.

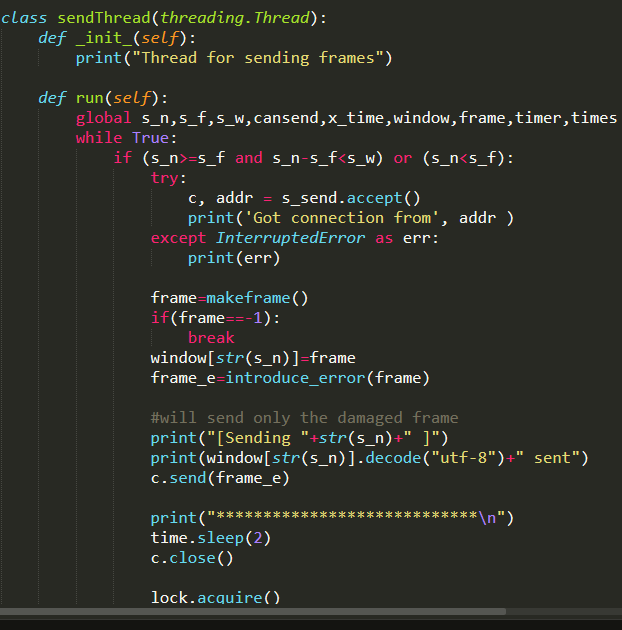


* + Resends only the damaged /NAK/ lost frame from s\_f to s\_n in case of timeout in a new thread at the end.

****

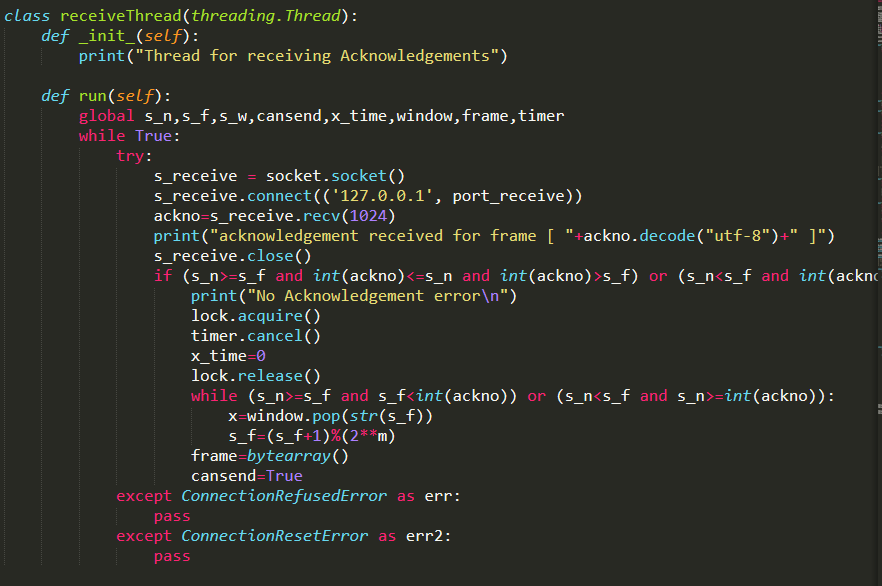
* + **Threading class**
* **class sending\_thread(threading.Thread):**

This class handles the sending of the data frames from the sender side to the receiver side. The **while()** loop runs infinitely to run the send operation whenever frames are ready and acknowledgment is received. When the sender is ready to send and there is a request to send, the sending socket checks for a connection with receiver port. If the connection is found sender creates a new frame from the dataword and introduces error (if any) to it. Then the sender sends the frame to the receiver port using the sending socket and the timer starts and cansend is turned false.

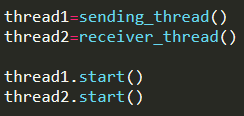


* **class receiver\_thread(threading.Thread)**

This class handles the receiving of the acknowledgement from the receiver side. The receiver socket searches for a connection from the sending socket in the receiver side. If there is any connection, the socket accepts the acknowledgment. If the acknowledgement is not corrupted and it is same or greater than s\_f, then the timer is stopped and cansend is turned true. Again, all the frames from s\_f to the acknowledgement number is purged.

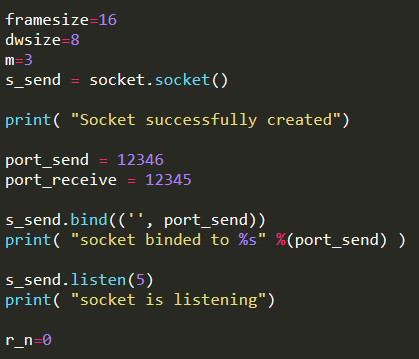
****

* + **Main Thread**

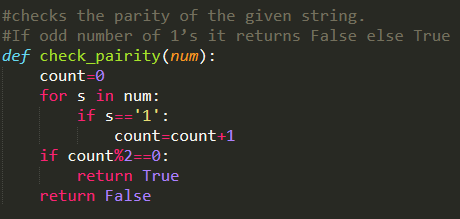
****

* + *gbn\_receiver.py*

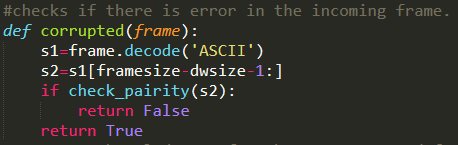
Global variables and modules



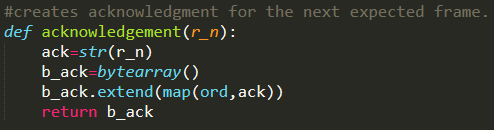
* This method checks the parity of the given string. If the odd number of 1’s, it returns False else True.



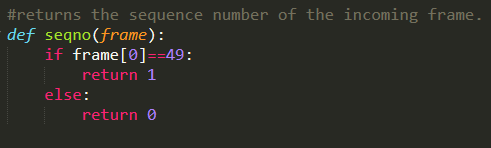
* This method checks whether there is an error in an incoming frame.



* This method creates the ACK for the next expected frame.



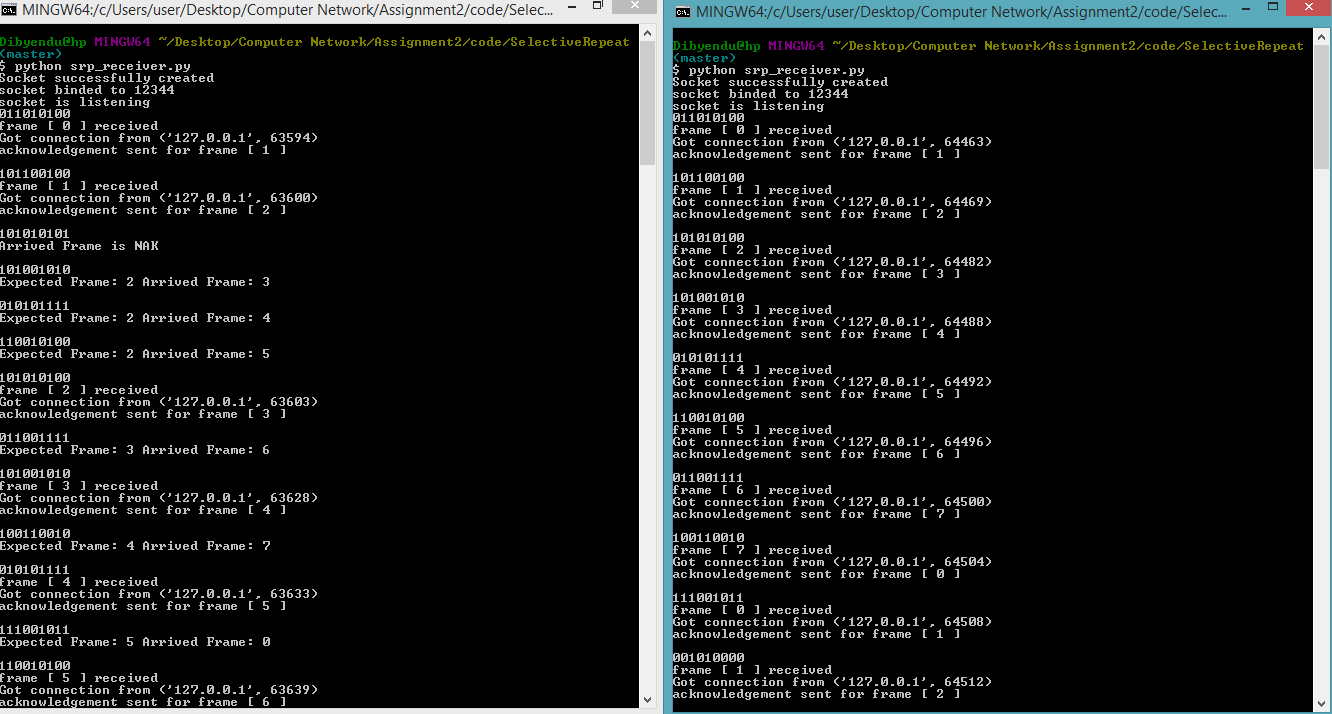
* This function defines the sequence number of the incoming frame.



* Main Thread

****

* **OUTPUT**

****

* + **Result** :

The performance of the above algorithm is measured in terms of throughput. The entire data file consists of 20 datawords. So for sending 20 frames in average of 65 attempts were taken considering insertion of random errors. Considering timeout after 5 seconds average propagation delay is 2 seconds.

## Analysis:

The program may have some possible bugs due to the lack of randomness in injecting the error. Again, the program is implemented for one sender and receiver that can be extended up to multiple senders and receivers. The last few bits from the input file is discarded to make the dataword size same. This bug can be overcome by padding of 0’s or 1’s.

## Comment:

The assignment was very helpful in better understanding of the implementation of one of the most popular flow control algorithms in noisy channel. Its difficulty level can be rated as extremely high.