

Computer Networks

Assignment-1

November 02, 2020

Overview


The main goal of the assignment is to simulate the different error detection techniques namely CRC(Cyclic Redundancy check), VRC(Vertical Redundancy Check), LRC(Longitudinal Redundancy Check) and CheckSum.

Goals

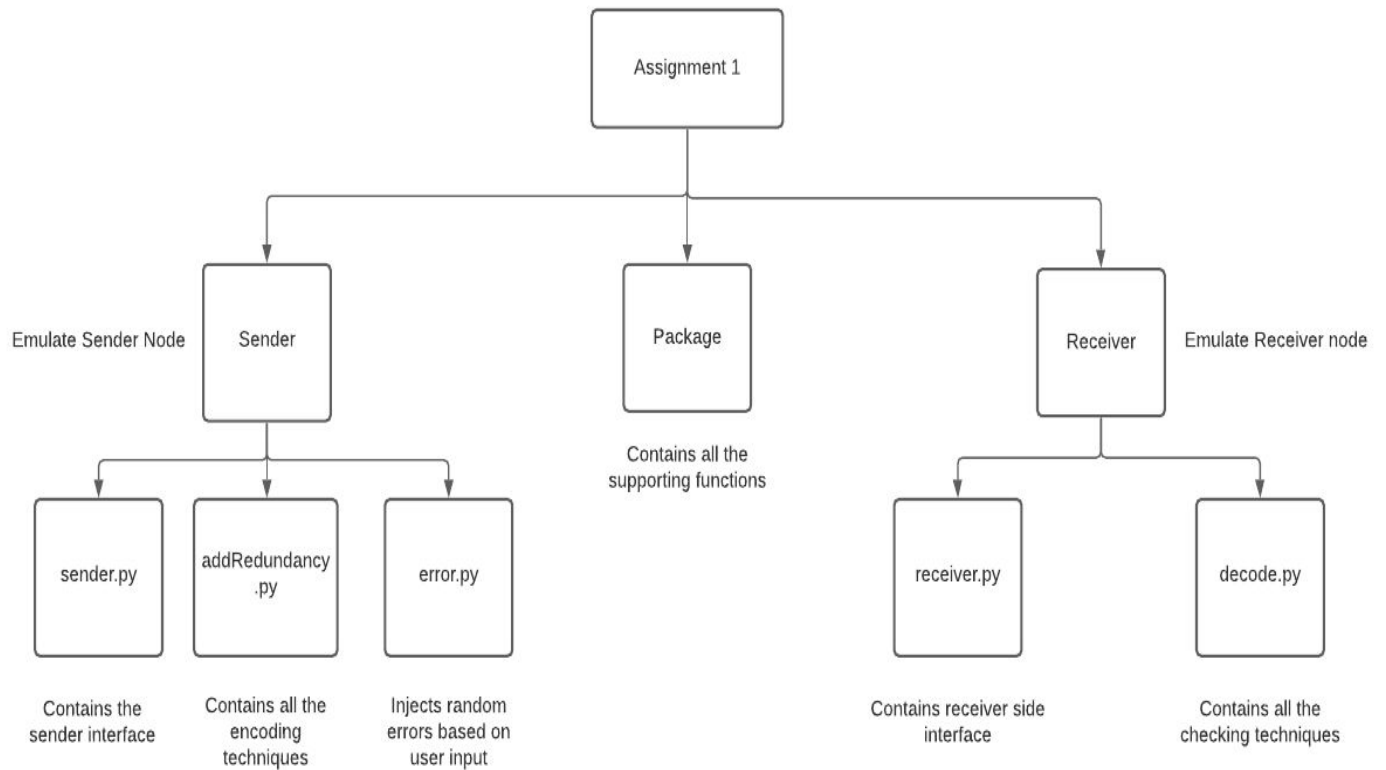
Implementing different Error detection schemas and test their relative productivity and accuracy.And Testing their working based on real life scenarios.

Specifications

The code has been written into **python**. I have used **socket programming** to simulate the real life scenarios of data transmission and error detection. I also have used normal **text files as input** files to imitate the real life scenarios as closely as possible. The errors are injected at a **random basis**.



FileStructure and Code



The sender directory imitates the sender node and the Receiver directory imitates the Receiver node. The input.txt file actually resides in the Sender Node. error.py file is responsible for injecting errors into the input file. I have taken the data frame as **8 bits (1 byte)** and encoded the frame with each error detection technique. The receiver node contains decode.py which checks all the output files and determines whether there are any discrepancies with the files or not.

Procedure

I have implemented such that it looks like the messages are getting transferred between sender and receiver node. The output looks like this. The receiver is kind of acting as a server here.

```

fish /home/soumalya/Desktop/MotherFolder/Assignment-5th...
> python3 sender.py
Enter the Filename: input.txt
Enter the name of the method!
* LRC
* CRC
* VRC
* Checksum
Enter the name: lrc
Generating redundancy using LRC technique
Output file "LRCOutput.dat" created!
LRC added to the text file!
Do you want to eject error?(y/n)
No error introduced!
*****MSG*****
No error detected!
File received successfully!
>

python3 receiver.py
Receiver is listening...
Connection created!
Output file LRCOutput.dat accepted!
Decoding output file with lrc method!
Decoding with LRC technique!
*****MSG*****
File received successfully!
Receiver is still listening...

```

Left side is Sender node and right side is receiver node.

After introducing random error to the channel. And also using checksum technique.

```

fish /home/soumalya/Desktop/MotherFolder/Assignment-5th...
> python3 sender.py
Enter the Filename: input.txt
Enter the name of the method!
* LRC
* CRC
* VRC
* Checksum
Enter the name: checksum
Generating redundancy using checksum technique
Output file "CHECKSUMOutput.dat" created!
Checksum added to the text file!
Do you want to eject error?(y/n)
Enter no of errors you want to introduce: 1
Error Introduced!
*****MSG*****
Corrupted File received!
Request for retransmission!
>

python3 receiver.py
Receiver is listening...
Connection created!
Output file CHECKSUMOutput.dat accepted!
Decoding output file with checksum method!
Decoding with CheckSum technique!
*****MSG*****
Corrupted File received!
Request for retransmission!
Receiver is still listening...

```

Left side is sender side and right side is receiver side

Observation

- I actually didn't have control over the random error injected by the program. So I actually couldn't be able to generate the scenarios mentioned in the question paper. But it can be done manually and can be proved that the question-mentioned scenarios are very possible. So it's always a good idea to encode the file with more than one error detection technique.

- The **checksum technique is very sensitive to errors**. So it's always a good idea to encode files with checksum. It has a failure rate of mere 0.5% apprx.
- The CRC technique is also very much reliable. It also can be used to detect real life data transfer errors.

Results

1. Using 4 error schemes guarantees errors are caught in **97-99% of the time**.
2. CRC 24 was better at checking errors than CRC ITU with CRC 24 failing **only around 2-3%** of the time while that value goes up as high as **8-10% when CRC ITU is used**.