

UNIVERSIDAD DEL VALLE DE GUATEMALA

CC3067 - Redes

Sección 10

Ing. Miguel Novella Linares



Laboratorio #2 - Esquemas de Deteccion y Correccion de Errores

Fernando José Garavito Ovando, 18071

Javier Andre Salazar de Leon 18764

Guatemala, 02 de agosto de 2022

Descripción de la práctica

La práctica se divide en 3 fases. En la primera fase tenemos el diseño del emisor y receptor. Donde el emisor tiene 4 capas que son aplicación, verificación, ruido y transmisión. La parte del receptor tiene 4 capas que son aplicación, verificación, codificación y transmisión. Luego se tiene que mandar información binaria a ASCII. En la capa de ruido, este debe ser para la cadena y con ello “voltear” los bits del bitarray. Tomando en cuenta que debe llegar con el receptor. La segunda fase se refiere a la implementación de algoritmos que son de detección y corrección. La tercera y última fase son de pruebas donde se puedan apreciar incluso resultados como las gráficas.

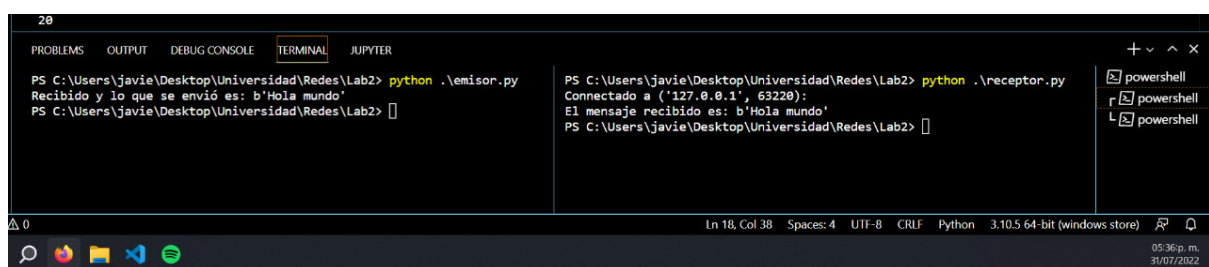
Descripción de los algoritmos utilizados

Algoritmo de Detección (Checksum): El siguiente algoritmo se refiere a un valor calculado que permite ver la validez de algo. En este caso se utilizan como transmisión de datos. Esta funciona en la comparación de un archivo con la que se proporciona con el archivo de origen.

Algoritmo de Detección (Hamming): Permite corregir errores que puedan suceder es trasladada desde el remitente al receptor. Se ingresan los datos que serán enviados, luego se calcula el número de bits de redundancia requeridos, se determina el bit/s de paridad, luego con un dato que dará negativo se testea la función.

Resultados (tablas, gráficas, etc.)

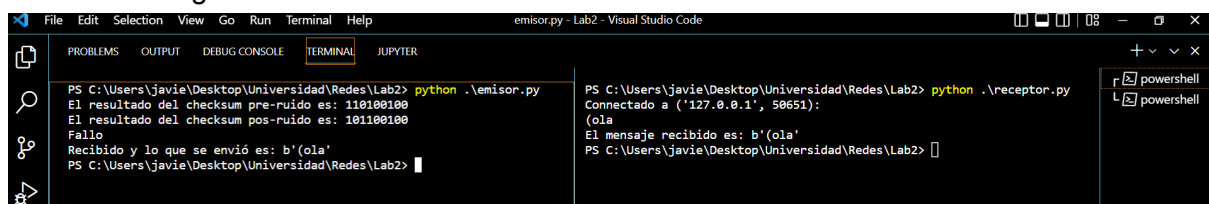
Conexión emisor y receptor:



```
PS C:\Users\javie\Desktop\Universidad\Redes\Lab2> python .\emisor.py
Recibido y lo que se envió es: b'Hola mundo'
PS C:\Users\javie\Desktop\Universidad\Redes\Lab2>

PS C:\Users\javie\Desktop\Universidad\Redes\Lab2> python .\receptor.py
Conectado a ('127.0.0.1', 63220):
El mensaje recibido es: b'Hola mundo'
PS C:\Users\javie\Desktop\Universidad\Redes\Lab2>
```

Pruebas del algoritmo Checksum:



```
PS C:\Users\javie\Desktop\Universidad\Redes\Lab2> python .\emisor.py
El resultado del checksum pre-ruido es: 110100100
El resultado del checksum pos-ruido es: 101100100
Fallo
Recibido y lo que se envió es: b'(ola'
PS C:\Users\javie\Desktop\Universidad\Redes\Lab2>

PS C:\Users\javie\Desktop\Universidad\Redes\Lab2> python .\receptor.py
Conectado a ('127.0.0.1', 50651):
(ola
El mensaje recibido es: b'(ola'
PS C:\Users\javie\Desktop\Universidad\Redes\Lab2>
```

```
emisor.py - Lab2 - Visual Studio Code

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

PS C:\Users\javie\Desktop\Universidad\Redes\Lab2> python .\emisor.py
El resultado del checksum pre-ruido es: 110100100
El resultado del checksum pos-ruido es: 101100100
Fallo
Recibido y lo que se envió es: b'ola'
PS C:\Users\javie\Desktop\Universidad\Redes\Lab2> python .\emisor.py
El resultado del checksum pre-ruido es: 110100100
El resultado del checksum pos-ruido es: 110100100
Exitoso
Recibido y lo que se envió es: b'hola'
PS C:\Users\javie\Desktop\Universidad\Redes\Lab2>

PS C:\Users\javie\Desktop\Universidad\Redes\Lab2> python .\receptor.py
Conectado a ('127.0.0.1', 50651):
ola
El mensaje recibido es: b'ola'
PS C:\Users\javie\Desktop\Universidad\Redes\Lab2> python .\receptor.py
Conectado a ('127.0.0.1', 50652):
hola
El mensaje recibido es: b'hola'
PS C:\Users\javie\Desktop\Universidad\Redes\Lab2>
```

Pruebas del algoritmo Hamming:

Discusión de los Resultados

Durante la práctica observamos que para los casos aislados y en el entorno que se probó, resulta más sencillo y conveniente utilizar un algoritmo de detección. Ya que los de corrección requieren de una mayor redundancia en los datos. Además de que comparando los algoritmos pudimos notar que CRC32 puede llegar a ser más robusto, ya que se puede usar tanto para el emisor como el receptor. Y pudimos investigar sobre la diferencia entre los códigos convolucionales respecto a los códigos de bloque lineal, y pudimos notar que los convolucionales resultan más adecuados para errores en rafaga, por eso resultan mayormente utilizados en satélites, reconocimiento de adn, etc.

Comentario en parejas

Javier Salazar: Pienso que el laboratorio tuvo un nivel un poco más alto del que esperaba, me resultó un poco complicado encontrar la documentación necesaria para hacer el laboratorio, pero una vez que la encontré no tomó tanto tiempo resolver los problemas.

Fernando Garavito: El laboratorio en general me pareció muy complicado, debido a que los algoritmos como tales no son usados con frecuencia y esto también aplica para la parte binaria y ASCII. Sin embargo es un laboratorio de mucho aprendizaje y muy útil.

Conclusiones

En general resultó ser una buena práctica y aprendimos mucho sobre los métodos de verificación y corrección de errores más utilizados hace un tiempo. Así como lo que se usa hoy en día, esto nos permitió ver lo complicado que resultaba antes debido a las limitaciones en el tamaño de las cadenas de mensajes.

Referencias

Python. Bitarray. Extraído de: <https://pypi.org/project/bitarray/>

Python. socket — Low-level networking interface. Extraído de: <https://docs.python.org/3/library/socket.html>

Implementing Checksum Using Python. Extraído de: <https://www.geeksforgeeks.org/implementing-checksum-using-python/>

Lifewire. What Is a Checksum?. Extraído de: <https://www.lifewire.com/what-does-checksum-mean-2625825>