EAC II

Errores en un canal de transmisión digital

13 de octubre de 2011

Autores:

Alan Pomerantz	Legajo: *51233*
Agustina Fainguersch	Legajo: *50589*
Tomás Mehdi	Legajo: *51014*

Mail:

tomymehdi@gmail.com
tmehdi@alu.itba.edu.ar

Índice

1. Enunciado

Los símbolos binarios **0** y **1** se transmiten a través de un canal de comunicación y se reciben correctamente con probabilidades $1 - p_0$ y $1 - p_1$ respectivamente como se describe en la figura. Los errores cometidos en la transmisión de símbolos diferentes se consideran independientes.

La información se transmite por paquetes de longitud $n = 256$ bits. Suponga que $p_0 = 0{,}04$, $p_1 = 0{,}06$ y que los símbolos binarios **0** y **1** aparecen con igual frecuencia.

2. Ejercicio 1

2.1. Enunciado

Mediante una simulación basada en considerar el envío de 1000 paquetes de longitud n estime la probabilidad de recibir paquetes con:

- ningún error
- exactamente un error
- exactamente dos errores

2.2. Tabla de errores en la simulación enviando mil paquetes

Cantidad de paquetes	Cantidad de errores
0	0
0	1
0	2
1	3
2	4
5	5
17	6
25	7
48	8
67	9
96	10
104	11
114	12
116	13
106	14
80	15
70	16
51	17
44	18
30	19
15	20
2	21
4	22
1	23
0	24
2	25
0	26
0	27
0	28
0	29
0	30

El resto de los errores tienen cero paquetes.

2.3. Tabla de errores en la simulación enviando diez mil paquetes

Cantidad de paquetes	Cantidad de errores
0	0
0	1
0	2
6	3
29	4
62	5
137	6
270	7
476	8
719	9
924	10
1063	11
1138	12
1163	13
1031	14
873	15
671	16
532	17
329	18
244	19
154	20
84	21
41	22
28	23
17	24
5	25
3	26
0	27
1	28
0	29
0	30

El resto de los errores tienen cero paquetes.

2.4. Estimación

Utilizando los resultados de la tabla resolvemos que la probabilidad de cada cantidad de errores es:

El valor que esta en la columna de 'Cantidad de paquetes' dividido 1000.

2.5. Calculo de la probabilidad de recibir paquetes sin error

Hacer un grafico de el arbol de probabilidad de 1 bit.
Probabilidad de que un bit sea correctamente recibido.

$$P((\text{envio } 0 \cap \text{llega } 0) \cup (\text{envio } 1 \cap \text{llega } 1)) =$$
$$P(\text{envio } 0 \cap \text{llega } 0) + (\text{envio } 1 \cap \text{llega } 1) =$$
$$P(\text{llega } 0|\text{envio } 0)P(\text{envio } 0) + P(\text{llega } 1|\text{envio } 1)P(\text{envio } 1) =$$
$$0{,}96 * 0{,}5 + 0{,}94 * 0{,}5 = 0{,}95$$

Al ser la probabilidad de cada bit independiente la probabilidad de enviar un paquete y que sea recibido correctamente es $0{,}95^n$. En este caso $n = 256$.

$$0{,}95^{256} \approx 1{,}98 * 10^{-6}$$

Siendo X = probabilidad de recibir sin errores un paquete de 1000. Sabemos que $X \sim \text{N}(0{,}95^{256}, 1000)$.

$$\binom{1000}{0} * (0{,}95^{256})^{1000} * (1 - 0{,}95^{256})^0 \approx 1{,}75 * 10^{-5703}$$

2.6. Comparación de la simulación con el cálculo del caso sin error

El calculo nos dio un numero insignificante y en la simulación de 1000 paquetes no hubo ningun paquete con cero, uno o dos errores. Se corrio 10 veces la simulación de los 1000 paquetes y en ninguna hubo casos en los que aparecieran paquetes con cero errores.

3. Ejercicio 2

3.1. Enunciado

Para mejorar la *confiabilidad* cada símbolo se transmite tres veces y el símbolo recibido se decodifica por la *regla de la mayoría*. Si se transmite un **0**(ó **1**) sí y solo sí la cadena de tres símbolos recibidos contiene al menos dos **0** (o **1** respectivamente). Cada paquete tiene

ahora $3n$ bits de longitud. Como antes estime la probabilidad de recibir paquete con:

- ningún error
- exactamente un error
- exactamente dos errores

3.2. Tabla de errores en la simulación del envio de mil paquetes

Cantidad de paquetes	Cantidad de errores
155	0
293	1
278	2
166	3
71	4
27	5
6	5
7	4
8	1
9	0
10	0

3.3. Tabla de errores en la simulación del envio de diez mil paquetes

Cantidad de paquetes	Cantidad de errores
1460	0
2815	1
2689	2
1761	3
823	4
322	5
97	5
24	4
8	1
1	0
0	0

El resto de los errores tienen cero paquetes.

3.4. Estimación

Utilizando los resultados de la tabla resolvemos que la probabilidad de cada cantidad de errores es aproximadamente:

El valor que esta en la columna de 'Cantidad de paquetes' dividido 1000.

4. Ejercicio 3

4.1. Enunciado

Un sistema más simple de detección de errores es el denominado *Checksum*, ó de suma de comprobación. Se añade al final del bloque de datos transmitidos la suma de todos ellos. El receptor debe comprobar que este dato se corresponde efectivamente con la suma de los datos recibidos. De no se así, es que ha ocurrido un error, por lo que debe solicitar al transmisor que repita la transmisión del bloque de datos.

Suponga que al bloque de 256 bits se le añade un byte (8 bits) con la suma de los dígitos del bloque en binario. El bloque completo de 264 bits se transmite por el canal.

Mediante una simulación basada en considerar el envio de 1000 paquetes de longitud 264 bits estime las probabilidades de:

- que el receptor solicite el reenvio del paquete(por no haber coincidido la suma de los 256 primeros bits del bloque con la suma almacenada en los 8 bits de checksum).
- que el mensaje de 256 bits haya sido recibido con errores sabiendo que el receptor solicitó el reenvío.

4.2. Estimación de solicitar el reenvio

Luego de correr la simulación con los 1000 paquetes la cantidad que se reciben con error es de 66. Lo que nos permite decir que la probabilidad aproximada de que se solicite el reenvio es de $\frac{66}{1000}$

4.3. Estimación de error en el reenvio

Hacer un grafico de el arbol de probabilidad de.

Siendo $P(\mathrm{C}')$ la probabilidad de recibir incorrectamente y $P(\hat{\mathrm{C}}')$ la probabilidad estimada de recibir incorrectamente. Lo que debemos estimar es:

$$P(\hat{\mathrm{C}}' \cap \hat{\mathrm{C}}') = P(\hat{\mathrm{C}}' | \hat{\mathrm{C}}') P(\hat{\mathrm{C}}')$$

y al ser sucesos independientes

$$P(\hat{C}'|\hat{C}') = P(\hat{C}')$$

Entonces

$$P(\hat{C}' \cap \hat{C}') = P(\hat{C}')^2$$

5. Códigos en java y documentaciones externas

5.1. random

public static double random()

Returns a double value with a positive sign, greater than or equal to 0,0 and less than 1,0. Returned values are chosen pseudorandomly with (approximately) uniform distribution from that range. When this method is first called, it creates a single new pseudorandom-number generator, exactly as if by the expression *new java.util.Random*.

This new pseudorandom-number generator is used thereafter for all calls to this method and is used nowhere else. This method is properly synchronized to allow correct use by more than one thread. However, if many threads need to generate pseudorandom numbers at a great rate, it may reduce contention for each thread to have its own pseudorandom-number generator.

Returns a pseudorandom double greater than or equal to 0.0 and less than 1,0.

5.2. Código en Java para la simulación y obtencion de datos

```
/*
 * Clase para simular un paquete de n bits.
 */
public class Paquete {
    public static final int n = 256;
    private int bits[] = new int[n];

    public int[] getBits() {
        return bits;
    }

    @Override
    public String toString() {
        String s = "{";
        for (int i = 0; i < n - 1; i++) {
            s += bits[i] + ", ";
        }
        s += bits[n - 1];
        s += "}";
        return s;
    }

    public void inicializar() {
        int num = 0;
        for (int i = 0; i < n; i++) {
            if (Math.random() >= 0.5) {
                num = 1;
            } else {
                num = 0;
            }
            bits[i] = num;
        }
    }
```

```
}
```

```
public class SimulacionEj1 {

    private static final int cantPaquetes = 1000;

    public static void main(String[] args) {
        /*
         * Creacion de paquetes
         */
        Paquete paquetes[] = new Paquete[cantPaquetes];
        for (int i = 0; i < cantPaquetes; i++) {
            paquetes[i] = new Paquete();
            paquetes[i].inicializar();
        }
        /*
         * Envio de paquetes
         */
        Paquete paquetesRecividos[] = new Paquete[cantPaquetes];
        for (int i = 0; i < cantPaquetes; i++) {
            paquetesRecividos[i] = enviar(paquetes[i]);
        }

        /*
         * Control de errores
         */
        int[] cantErrores = new int[Paquete.n];
        int countErrores;
        for (int i = 0; i < cantPaquetes; i++) {
            countErrores = 0;
            for (int j = 0; j < Paquete.n; j++) {
                if (paquetes[i].getBits()[j] != paquetesRecividos[i].↩
                    getBits()[j]) {
                    countErrores++;
                }
            }
            cantErrores[countErrores]++;
        }

        /*
         * Impresion de un vector con la cantidad de errores. En la ↩
             posicion 0 la cantidad
         * de paquetes con cero errores, en la posicion 1 la cantidad ↩
             de paquetes con 1 error, ...
         */
        System.out.print("{");
        for (int i = 0; i < Paquete.n-1; i++) {
            System.out.print(cantErrores[i] + ", ");
        }
        System.out.print(cantErrores[Paquete.n-1]);
        System.out.print("}");
    }


    /*
     * Mensaje que modifica un paquete segun el valor obtenido por ↩
         Math.random()
     */
    private static Paquete enviar(Paquete paquete) {
        Paquete resp = new Paquete();
        double rand;
        for (int i = 0; i < Paquete.n; i++) {
            rand = Math.random();
            if ((paquete.getBits()[i] == 0 && rand >= 0.96)
                    || (paquete.getBits()[i] == 1 && rand < 0.94)) {
                resp.getBits()[i] = 1;
            } else {
                resp.getBits()[i] = 0;
            }
        }
        return resp;
```

```
    }
}
```

```
public class PaqueteMultiple {
    static final int n = 3;
    private Paquete paquetes[] = new Paquete[3];

    public PaqueteMultiple() {
        for(int i = 0 ; i<n ; i++){
            paquetes[i] = new Paquete();
        }
    }

    public void inicializar() {
        for (int i = 0; i < n; i++) {
            paquetes[i] = new Paquete();
            paquetes[i].inicializar();
        }
    }

    @Override
    public String toString() {
        String s = "";
        for (int i = 0; i < n; i++) {
            s += paquetes[i].toString();
            s += "\n";
        }
        return s;
    }

    public int[] getBits(int j) {
        return paquetes[j].getBits();
    }

    public int getRealBit(int j) {
        int sum = 0;
        for (int i = 0; i < n; i++) {
            sum += paquetes[i].getBits()[j];
        }
        if (sum >= (n/2)+1) {
            return 1;
        }
        return 0;
    }

}
```

```
public class SimulacionEj2 {
    private static final int cantPaquetes = 1000;

    public static void main(String[] args) {
        /*
         * Creacion de paquetes
         */
        Paquete paquetes[] = new Paquete[cantPaquetes];
        for (int i = 0; i < cantPaquetes; i++) {
            paquetes[i] = new Paquete();
            paquetes[i].inicializar();
        }
        /*
         * Envio de paquetes
         */
        PaqueteMultiple paquetesRecividos[] = new PaqueteMultiple[↩
            cantPaquetes];
        for (int i = 0; i < cantPaquetes; i++) {
            paquetesRecividos[i] = enviar(paquetes[i]);
        }

        /*
```

```
         * Creacion de los paquetes con los multipaquetes recibidos
         */
        Paquete paquetesR[] = new Paquete[cantPaquetes];
        for (int i = 0; i < cantPaquetes; i++) {
            paquetesR[i] = new Paquete();
            for (int j = 0; j < Paquete.n; j++) {
                paquetesR[i].getBits()[j] = paquetesRecividos[i].↩
                    getRealBit(j);
            }
        }

        /*
         * Control de errores
         */
        int[] cantErrores = new int[Paquete.n];
        int countErrores;
        for (int i = 0; i < cantPaquetes; i++) {
            countErrores = 0;
            for (int j = 0; j < Paquete.n; j++) {
                if (paquetes[i].getBits()[j] != paquetesR[i].getBits()↩
                    [j]) {
                    countErrores++;
                }
            }
            cantErrores[countErrores]++;
        }

        /*
         * Impresion de un vector con la cantidad de errores. En la ↩
           posicion 0
         * la cantidad de paquetes con cero errores, en la posicion 1 ↩
           la
         * cantidad de paquetes con 1 error, ...
         */
        System.out.print("{");
        for (int i = 0; i < Paquete.n - 1; i++) {
            System.out.print(cantErrores[i] + ", ");
        }
        System.out.print(cantErrores[Paquete.n - 1]);
        System.out.print("}");
    }

    /*
     * Mensaje que modifica un paquete segun el valor obtenido por ↩
       Math.random()
     */
    private static PaqueteMultiple enviar(Paquete paquete) {
        PaqueteMultiple resp = new PaqueteMultiple();
        double rand;
            for (int i = 0; i < Paquete.n; i++) {
                for (int j = 0; j < PaqueteMultiple.n; j++) {
                rand = Math.random();
                if ((paquete.getBits()[i] == 0 && rand >= 0.96)
                        || (paquete.getBits()[i] == 1 && rand < 0.94))↩
                            {
                    resp.getBits(j)[i] = 1;
                } else {
                    resp.getBits(j)[i] = 0;
                }
            }
        }
        return resp;
    }
}
```

```
public class PaqueteConChecksum {
    public static final int n = 264;
    private int bits[] = new int[n];

    public int[] getBits() {
        return bits;
```

```
    }

    @Override
    public String toString() {
        String s = "{";
        for (int i = 0; i < n - 1; i++) {
            s += bits[i] + ", ";
        }
        s += bits[n - 1];
        s += "}";
        return s;
    }

    public void inicializar() {
        int num = 0;
        int checkSum = 0;
        for (int i = 0; i < n - 8; i++) {
            if (Math.random() >= 0.5) {
                num = 1;
                checkSum++;
            } else {
                num = 0;
            }
            bits[i] = num;
        }
        String binario = Integer.toBinaryString(checkSum);
        for (int i = n - binario.length(); i < n; i++) {
            bits[i] = binario.charAt(i - n + binario.length()) - '0';
        }
    }

    public boolean tieneError() {
        int sum = 0;
        for (int i = 0; i < n - 8; i++) {
            sum += bits[i];
        }
        String s = "";
        int checkSum = 0;
        for (int i = n - 8; i < n; i++) {
            s += Integer.toString(bits[i]);
        }
        checkSum = Integer.parseInt(s, 2);
        if (sum == checkSum) {
            return true;
        }
        return false;
    }

    @Override
    public boolean equals(Object obj) {
        if(!(obj instanceof PaqueteConChecksum)){
            return false;
        }
        PaqueteConChecksum paquete = (PaqueteConChecksum)obj;
        for(int i = 0; i <n ; i++){
            if( paquete.getBits()[i] != bits[i]){
                return false;
            }
        }
        return true;
    }
}
```

```
public class SimulacionEj3 {

    private static final int cantPaquetes = 1000;

    public static void main(String[] args) {
        /*
         * Creacion de paquetes
         */
```

```
        PaqueteConChecksum paquetes[] = new PaqueteConChecksum[↩
            cantPaquetes];
        for (int i = 0; i < cantPaquetes; i++) {
            paquetes[i] = new PaqueteConChecksum();
            paquetes[i].inicializar();
        }
        /*
         * Envio de paquetes
         */
        PaqueteConChecksum paquetesRecividos[] = new ↩
            PaqueteConChecksum[cantPaquetes];
        for (int i = 0; i < cantPaquetes; i++) {
            paquetesRecividos[i] = enviar(paquetes[i]);
        }

        /*
         * Control de errores
         */
        int cantErroresEnLos1000Paquetes = 0;
        for (int i = 0; i < cantPaquetes; i++) {
            if (paquetesRecividos[i].tieneError()) {
                cantErroresEnLos1000Paquetes++;
            }
        }
        int cantPaquetesErroneosPeroCorrectosPorElChecksum = 0;
        for (int i = 0; i < cantPaquetes; i++) {
            if (!paquetes[i].equals(paquetesRecividos)) {
                cantPaquetesErroneosPeroCorrectosPorElChecksum++;
            }
        }

        /*
         * Impresion de la cantidad de paquetes que tienene errores ↩
             segun el checksum
         */
        System.out
                .println("Cantidad de paquetes erroneos de los 1000 ↩
                    segun el checksum: "
                        + cantErroresEnLos1000Paquetes);
        /*
         * Impresion de la cantidad de paquetes que tienen cero ↩
             errores al ser recividos
         */
        System.out.println("Cantidad de paquetes realmente erroneos: "
                + cantPaquetesErroneosPeroCorrectosPorElChecksum);
    }

    /*
     * Mensaje que modifica un paquete segun el valor obtenido por ↩
         Math.random()
     */
    private static PaqueteConChecksum enviar(PaqueteConChecksum ↩
        paquete) {
        PaqueteConChecksum resp = new PaqueteConChecksum();
        double rand;
        for (int i = 0; i < PaqueteConChecksum.n; i++) {
            rand = Math.random();
            if ((paquete.getBits()[i] == 0 && rand >= 0.96)
                    || (paquete.getBits()[i] == 1 && rand < 0.94)) {
                resp.getBits()[i] = 1;
            } else {
                resp.getBits()[i] = 0;
            }
        }
        return resp;
    }
}
```