

Práctica 5 - Simulación de protocolos de control de acceso al medio ALOHA

1. Introducción

Los protocolos de control de acceso al medio permiten la compartición de un medio físico para la transmisión de múltiples fuentes. En estos medios compartidos la transmisión simultánea de dos o más fuentes provoca lo que se vienen a llamar "colisiones", durante las cuales la suma de las señales de las diferentes fuentes hace que un receptor no pueda recibir correctamente ninguna de ellas. Un protocolo de control de acceso al medio (MAC) para una tecnología de paquetes va a definir la forma de alternar el uso del medio compartido para lograr envíos con éxito de estos paquetes.

Según el protocolo de control de acceso al medio podremos obtener mejor o peor rendimiento del sistema de comunicaciones con medio compartido, el cual podremos medir mediante la probabilidad de que un paquete colisione, o con el tiempo hasta que un paquete logre atravesar el medio con éxito (suponiendo que hacemos retransmisiones).

Algunos protocolos de control de acceso al medio permiten calcular una solución analítica con unas hipótesis más o menos razonables y herramientas matemáticas sencillas, mientras que otros se vuelven rápidamente problemas complejos o incluso intratables matemáticamente. Una alternativa para analizar el comportamiento del sistema es ponerlo en funcionamiento, lo cual claramente en las primeras etapas del desarrollo de un protocolo no es muy práctico. Otra alternativa es simular el comportamiento del sistema mediante un programa informático. Eso quiere decir simular los sucesos que se producirían en el sistema real (comienzo de un envío por una estación, finalización, comienzo de otro envío, comienzo de una colisión, etc). Mediante esta simulación podemos hacer una estimación de esos resultados de rendimiento (tiempo hasta que un paquete atraviese el sistema, probabilidad de que colisione, etc).

En este ejercicio se propone desarrollar pequeños simuladores para estimar resultados reales o verificar resultados analíticos. Podremos por ejemplo cambiar condiciones del funcionamiento del sistema que lleven a un difícil cálculo analítico pero que se puedan simular con sencillez.

Tenga en cuenta que necesitará realizar simulaciones donde el número de paquetes puede ser arbitrariamente elevado, por lo que debe evitar generar y almacenar todas las llegadas en memoria.

2. ALOHA puro

Cree un programa en Java para simular el comportamiento de un enlace ALOHA (no ranurado). Los parámetros del programa serán:

- La tasa media de llegadas por unidad de tiempo del tráfico agregado al enlace.
- El tiempo de transmisión (constante) de los paquetes.
- El número de paquetes que se van a intentar enviar.
- Opcionalmente, la semilla o *seed*¹ utilizada para la inicialización del generador de números.

Las llegadas representan los instantes en los que las estaciones tienen un paquete para enviar y van a comenzar la transmisión. Supondremos que hay muchas estaciones, con poco tráfico cada una, con lo que es seguro suponer que dos transmisiones cercanas en el tiempo provienen siempre de estaciones diferentes y pueden colisionar.

Vamos a ignorar el mecanismo de confirmaciones. Supondremos que o bien no existen o simplemente algunos de esos paquetes representados por las llegadas son las confirmaciones. Ignoraremos también las retransmisiones, es decir, el proceso de instantes de llegadas representa todos los paquetes que se intentan enviar (paquetes de datos, confirmaciones, retransmisiones).

Las llegadas siguen un proceso de Poisson de tasa el primer parámetro del programa. Eso quiere decir que los tiempos entre llegadas siguen una distribución exponencial con ese mismo parámetro y son independientes. En Java es sencillo generar números aleatorios con una distribución uniforme utilizando la clase `Random`². Estos se pueden transformar de forma sencilla en otra distribución mediante la técnica de la transformada inversa³ y la utilización de las funciones disponibles en la clase `Math`⁴.

Para cada paquete debe calcular si colisiona o no y llevar una cuenta de cuántos paquetes colisionarían, con lo que podrá estimar la probabilidad de colisión.

Como salida del programa muestre, para cada paquete de la secuencia de tiempos, el número del paquete, el tiempo del evento de llegada, si colisiona o no y la estimación hasta el momento de esa probabilidad de colisión. Con esa salida podrá ir viendo si el resultado se va estabilizando alrededor de un valor a medida que avanza la simulación y la probabilidad se estima con más realizaciones del suceso de envío.

```
$ java AlohaPuro 0.5 0.25 10000 243305
1 0.223997 no-colisiona 0.000000
```

¹ https://en.wikipedia.org/wiki/Random_seed

² <https://docs.oracle.com/javase/8/docs/api/java/util/Random.html>

³ https://es.wikipedia.org/wiki/Distribuci%C3%B3n_exponencial#M%C3%A9todos_computacionales

⁴ <https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html>

```

2 0.765380 si-colisiona 0.500000
3 0.966814 si-colisiona 0.666667
4 1.597588 no-colisiona 0.500000
5 4.946889 no-colisiona 0.400000
...
1000 2009.230177 no-colisiona 0.221000
...
10000 20104.631175 no-colisiona 0.210000

```

Pruebe a representar la probabilidad de colisión obtenida mediante la simulación con el valor teórico utilizando gnuplot. Dese cuenta de que tendrá que ajustar el cálculo del valor teórico en función de los parámetros de la simulación.

```

$ java AlohaPuro 0.5 0.25 1000 243305 | gnuplot -p -e "plot '-' using 1:4,
1 -exp(-2 * 0.5 * 0.25)"
$ java AlohaPuro 0.5 1.00 1000 243305 | gnuplot -p -e "plot '-' using 1:4,
1 -exp(-2 * 0.5 * 1.00)"
$ java AlohaPuro 2.0 0.25 1000 243305 | gnuplot -p -e "plot '-' using 1:4,
1 -exp(-2 * 2.0 * 0.25)"

```

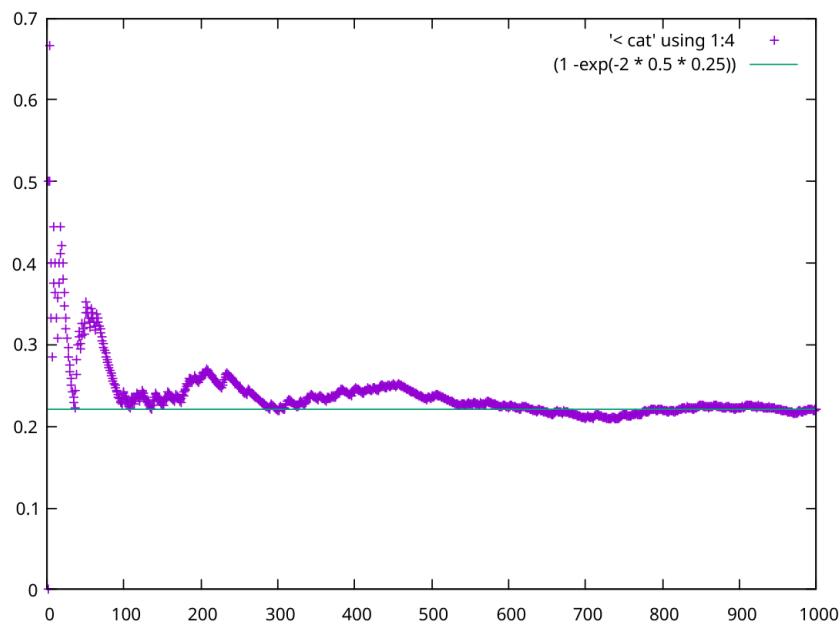


Figura 1 – Convergencia de la probabilidad de colisión.

Punto de control 1 (40%): Muestre el programa y el resultado de la simulación.

Nota: Conserve el código del programa actual al pasar al siguiente punto de control (dúpliquelo y edite la copia). Posteriormente se le pedirá volver a modificar este programa.

3. ALOHA: tiempos de servicio variables

Modifique el programa para que el tiempo de transmisión deje de ser constante. En concreto, el tiempo de transmisión deberá generarse de forma aleatoria siguiendo una distribución exponencial cuyo promedio sea el valor del parámetro indicado.

Con esta modificación se podrá ver si los resultados se parecen a lo que se obtenía con tiempos constantes. Con igual tasa media de llegadas e igual tiempo medio de servicio, la carga promedio ofrecida es la misma. Por lo tanto, intuitivamente podríamos esperar el mismo resultado de probabilidad de colisión. Compruébelo realizando los cambios y comparando el resultado de la simulación con el valor teórico anterior. ¿Qué está sucediendo?

```
$ java AlohaTiemposVariables 0.5 0.25 10000 243305 | gnuplot -p -e "plot '-' using 1:4, 1 -exp(-2 * 0.5 * 0.25), 1 -exp(-0.5 * 0.25) / (1 + 0.5 * 0.25)"
```

```
$ java AlohaTiemposVariables 0.5 1.00 10000 243305 | gnuplot -p -e "plot '-' using 1:4, 1 -exp(-2 * 0.5 * 1.00), 1 -exp(-0.5 * 1.00) / (1 + 0.5 * 1.00)"
```

```
$ java AlohaTiemposVariables 2.0 0.25 10000 243305 | gnuplot -p -e "plot '-' using 1:4, 1 -exp(-2 * 2.0 * 0.25), 1 -exp(-2.0 * 0.25) / (1 + 2.0 * 0.25)"
```

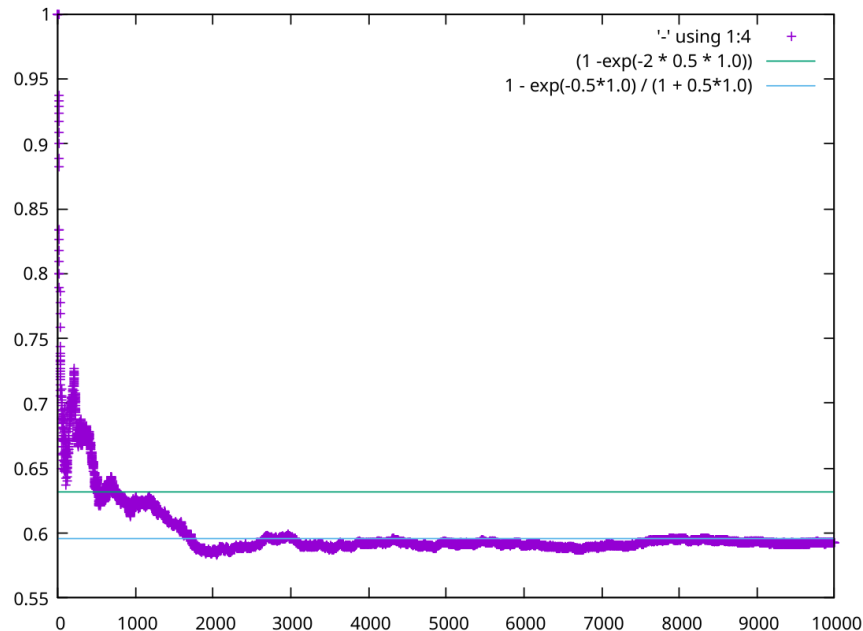


Figura 2 – Convergencia de la probabilidad de colisión con tamaños aleatorios exponencialmente distribuidos.

En vez de calcular una probabilidad de colisión promedio independiente del tamaño (usando “tamaño” como equivalente a tiempo de transmisión ya que suponemos tasa de transmisión constante), es posible calcular la probabilidad de

colisión de los paquetes en función de su tamaño, o una aproximación para cuando su tamaño está en un intervalo $[x, x+d)$. Estaríamos estimando una probabilidad condicionada al tamaño del paquete.

Modifique su programa para que calcule el ancho d de cada intervalo, de forma que se generen 100 intervalos equiespaciados que cubran aproximadamente el 99.9% de los tiempos de transmisión generados⁵. Imprima para cada paquete dos valores adicionales, el tamaño del paquete (discretizado al intervalo que corresponda) y la probabilidad de bloqueo exclusiva de ese intervalo.

Represente esta probabilidad de colisión frente al tamaño con el siguiente comando:

```
$ java AlohaTiemposVariables 0.5 0.25 100000 243305 | awk '{b[$5] = $6};  
END {for (x in b) print x, b[x]}' | gnuplot -p -e "set key bottom right;  
plot '-' using 1:2, 1 - exp(-0.5 * (0.25 + x))"  
$ java AlohaTiemposVariables 0.5 1.00 100000 243305 | awk '{b[$5] = $6};  
END {for (x in b) print x, b[x]}' | gnuplot -p -e "set key bottom right;  
plot '-' using 1:2, 1 - exp(-0.5 * (1.00 + x))"  
$ java AlohaTiemposVariables 2.0 0.25 100000 243305 | awk '{b[$5] = $6};  
END {for (x in b) print x, b[x]}' | gnuplot -p -e "set key bottom right;  
plot '-' using 1:2, 1 - exp(-2.0 * (0.25 + x))"
```

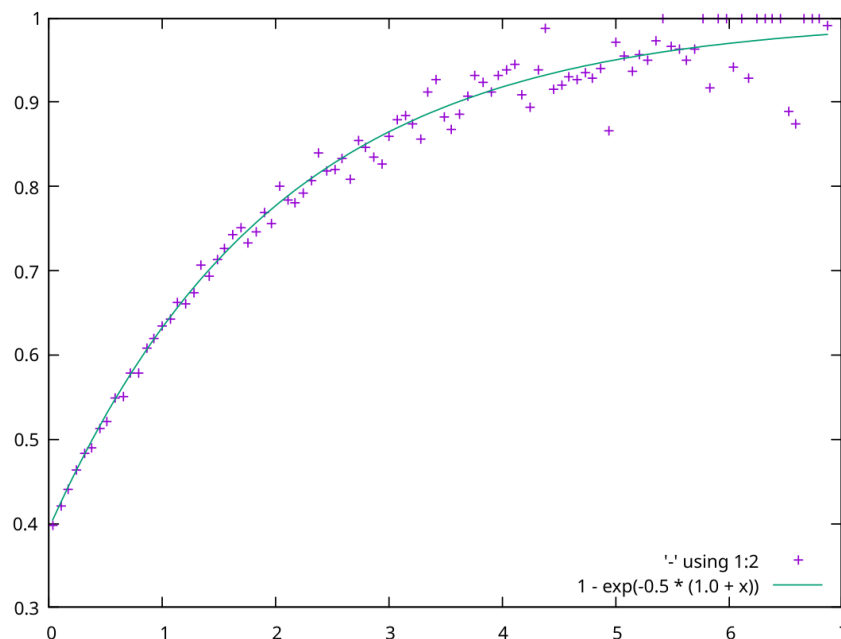


Figura 3 – Probabilidad de colisión frente al tamaño del paquete.

Punto de control 2 (40%): Muestre el programa y los resultados de las simulaciones realizadas.

⁵ https://es.wikipedia.org/wiki/Distribuci%C3%B3n_exponencial#Cuantiles

4. ALOHA ranurado

Vamos a comparar ahora el caso inicial de ALOHA con tiempos constantes frente a ALOHA ranurado. Los eventos de llegada ahora deberán por tanto esperar al inicio de la siguiente ranura para empezar a ser transmitidos. Los paquetes colisionarán sí y solo sí hay varios paquetes transmitidos en la misma ranura.

Reinterprete en este caso el segundo argumento para que indique el tamaño de cada ranura. Añada dos columnas adicionales a la salida: el índice de la ranura y el tiempo de inicio de la ranura.

```
$ java AlohaRanurado 0.5 0.25 10000 243305
1 0.223997 no-colisiona 0.000000 1 0.25
2 0.765380 si-colisiona 0.500000 4 1.00
3 0.966814 si-colisiona 0.666667 4 1.00
4 1.597588 no-colisiona 0.500000 7 1.75
5 4.946889 no-colisiona 0.400000 20 5.00
...
10000 20104.631175 no-colisiona 0.109500 80419 20104.75
```

Represente los valores de colisión obtenidos a partir de la simulación frente a los valores teóricos de ALOHA puro y ranurado.

```
$ java AlohaRanurado 0.5 0.25 10000 243305 | gnuplot -p -e "plot '-' using 1:4, 1 -exp(-2 * 0.5 * 0.25), 1 -exp(-0.5 * 0.25)"
$ java AlohaRanurado 0.5 1.00 10000 243305 | gnuplot -p -e "plot '-' using 1:4, 1 -exp(-2 * 0.5 * 1.00), 1 -exp(-0.5 * 1.00)"
$ java AlohaRanurado 2.0 0.25 10000 243305 | gnuplot -p -e "plot '-' using 1:4, 1 -exp(-2 * 2.0 * 0.25), 1 -exp(-2.0 * 0.25)"
```

Modifique el programa actual y el original (primer punto de control) para que al terminar impriman por la salida de error la intensidad de tráfico ofrecido y cursado:

```
$ java AlohaPuro 0.5 0.25 10000 243305 1>/dev/null
0.125000 0.098236
$ java AlohaRanurado 0.5 0.25 10000 243305 1>/dev/null
0.125000 0.110733
```

Cree un pequeño script en bash que ejecute ambos programas para distintos valores de carga ofrecida y acumule los resultados finales en un fichero de texto distinto para cada programa. Calcule para las cargas desde 0.05 hasta 4.0 Erlangs en incrementos de 0.05. Puede generar una lista de valores equiespaciados con el comando `seq <start> <step> <end>`. Si realiza la simulación con 10k paquetes y la semilla de los ejemplos deberá obtener los siguientes resultados:

```
$ cat sim_AlohaPuro.txt
0.050000 0.045398
```

```

0.100000 0.082648
0.150000 0.112198
...
4.000000 0.001194
$ cat sim_AlohaRanurado.txt
0.050000 0.047437
0.100000 0.090646
0.150000 0.130045
...
4.000000 0.067646

```

Represente con gnuplot la intensidad de tráfico cursado frente al tráfico ofrecido, junto con los valores teóricos de cada uno de los algoritmos de control de acceso al medio.

```

$ gnuplot -p -e "plot 'sim_AlohaPuro.txt' using 1:2,
'sim_AlohaRanurado.txt' using 1:2, x * exp(-x), x * exp(-2*x)"

```

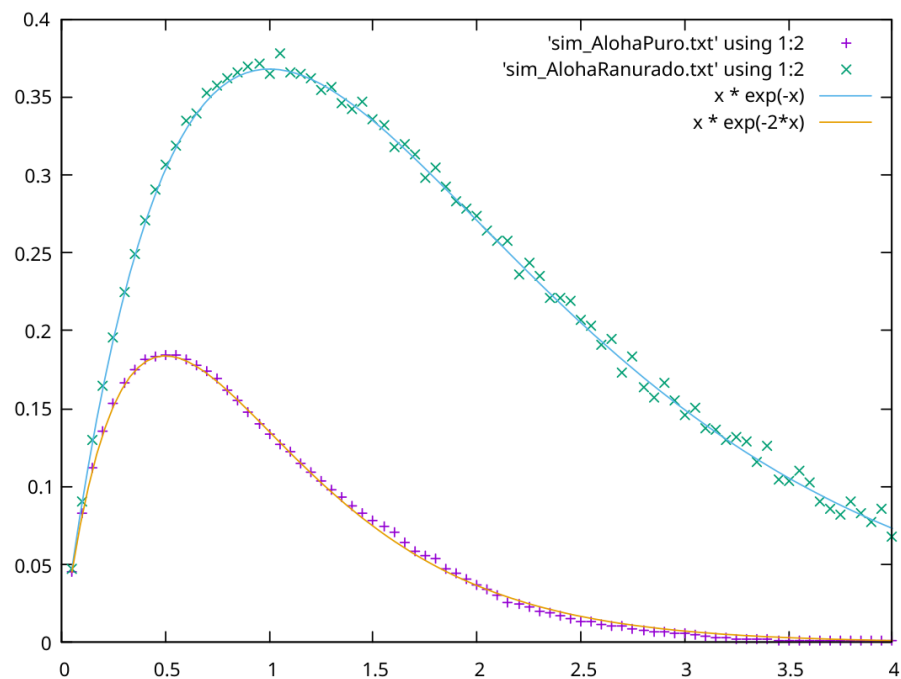


Figura 4 – Tráfico cursado frente al tráfico ofrecido. Comparación ALOHA puro y ALOHA ranurado

Punto de control 3 (20%): Muestre el programa y los resultados de las simulaciones realizadas.

5. ALOHA: confirmaciones y retransmisiones

Hemos supuesto que los ACKs y las retransmisiones eran parte de los intentos de envío que teníamos modelados en el proceso de llegadas. Ahora intentaremos introducir explícitamente las confirmaciones y retransmisiones en el modelo de simulación. Para ello, el proceso de llegadas representará solo los instantes en los que las estaciones intentan enviar un paquete nuevo. Por mantener la complejidad bajo control, asumiremos que el tiempo de propagación es despreciable.

La forma en la que se comportan las estaciones es la siguiente:

- Si un paquete no colisiona, el receptor deberá generar un paquete de confirmación (ACK), del mismo tamaño que el paquete original, en tiempo $t + T_x + D$. El tiempo D intenta modelar el retardo de procesamiento de la estación receptora, desde que termina de recibir el paquete hasta que lo procesa y decide que debe generar un ACK.
- Si la estación que ha transmitido el paquete de datos no obtiene confirmación (colisiona el paquete de datos o la confirmación), deberá comenzar a transmitir de nuevo el paquete en tiempo $t + 2T_x + 2D + M$. Así, está dejando el tiempo suficiente para que el paquete original se transmita, el receptor lo procese, se transmita el ACK y el mismo lo pueda procesar, más un tiempo adicional de margen M .
- Si tampoco recibe confirmación a la retransmisión, se descartará el paquete.

Pueden verse representados los distintos instantes de la línea temporal en la siguiente figura:

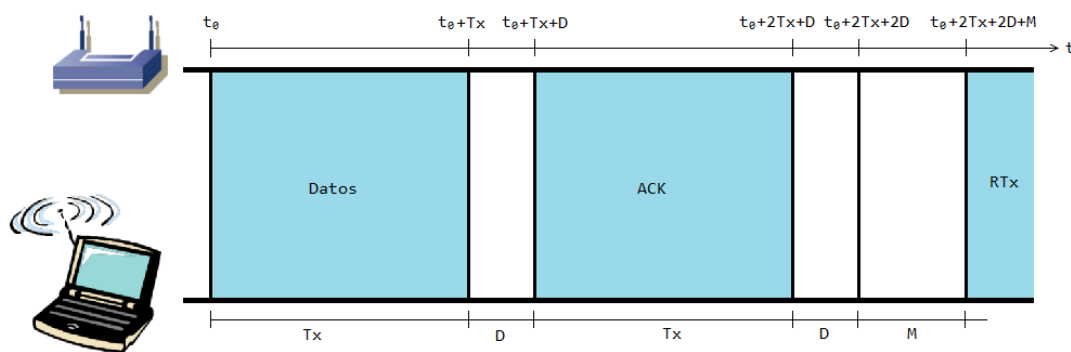


Figura 5: Línea temporal de las transmisiones de paquetes de datos, ACK y RTx.

A efectos de colisiones, los paquetes de confirmación y los retransmitidos deberán tratarse igual que las llegadas de nuevos paquetes. Es decir, estos pueden colisionar con cualquier otro paquete que se produzca dentro de su ventana de transmisión.

Para realizar esta implementación necesitara programar un simulador de eventos discretos⁶. Para ello, se define una cola de eventos ordenados temporalmente, que representan los instantes en los que puede ocurrir un cambio en el estado de la simulación. El programa deberá ir avanzando el tiempo de la simulación según extrae y procesa los distintos eventos de esta cola. Al procesar un evento, deberá introducir en la cola los nuevos eventos que se generen a causa del cambio del estado de la simulación.

Introduzca las siguientes clases en su programa para modelar los paquetes y los eventos del simulador.

```
enum TipoPaquete { DATOS, ACK }
class Paquete {
    final long id;
    final TipoPaquete tipo;
    final int rtxCount;
    boolean colisionado;

    public Paquete(long id, TipoPaquete tipo, int rtxCount) {
        this.id = id;
        this.tipo = tipo;
        this.rtxCount = rtxCount;
        this.colisionado = false;
    }
}

enum TipoEvento { INICIO, FIN }
class Evento implements Comparable {
    final TipoEvento tipo;
    final double tiempo;
    final Paquete paquete;

    public Evento(TipoEvento tipo, double tiempo, Paquete paquete) {
        this.tipo = tipo;
        this.tiempo = tiempo;
        this.paquete = paquete;
    }

    public int compareTo(Evento otro) {
        return Double.compare(this.tiempo, otro.tiempo);
    }
}
```

⁶ https://en.wikipedia.org/wiki/Discrete-event_simulation

Así, los paquetes se definirán con un id (número de llegada), un tipo (DATOS/ACK), el conteo de retransmisiones (0 para el primer intento, 1 para la retransmisión) y un flag que indica si ha colisionado o no.

Por el otro lado, los eventos se definen con un tipo (INICIO o FIN de transmisión), un tiempo en el que sucede y una referencia al paquete correspondiente. Estos eventos implementan los mecanismos necesarios para que sean comparables entre sí según el orden cronológico. De esta forma, cada transmisión de un paquete será representada por dos eventos, uno de inicio de la transmisión y otro de fin.

Así, su simulador podrá definir una cola de eventos futuros como una `PriorityQueue<Evento>`⁷ y almacenar en una lista todos los paquetes actualmente siendo transmitidos. Al procesar un evento de INICIO, si el número de paquetes en vuelo es superior a 1, todos los paquetes en vuelo deberán marcarse como colisionados. Al procesar un evento de FIN, deberá ver si el paquete ha colisionado o no y crear los eventos futuros que correspondan, así como incrementar la cuenta de paquetes colisionados.

Modifique su programa para realizar la simulación de eventos discretos. Tenga en cuenta que su programa deberá poder funcionar para un volumen elevado de llegadas, por lo que no es factible almacenarlas todas en memoria de forma simultánea. Añada el tipo de paquete y el conteo de retransmisión junto a la información que imprime para cada uno de ellos.

```
java AlohaAckRtx 0.5 0.25 0.05 0.05 10000 243305
1 0.223997 no-colisiona 0.000000 datos 0
1 0.523997 si-colisiona 0.500000 ack 0
2 0.765380 si-colisiona 0.666667 datos 0
1 0.873997 si-colisiona 0.750000 datos 1
3 0.966814 si-colisiona 0.800000 datos 0
...
10000 20104.631175 no-colisiona 0.377243 datos 0
10000 20104.931175 no-colisiona 0.377226 ack 0
```

Fíjese en que ahora la carga efectiva de tráfico sobre el sistema se ve incrementada con los paquetes de confirmación y las retransmisiones. Sin embargo, a la hora de calcular el tráfico cursado por la red, resulta más relevante conocer el porcentaje de llegadas que obtienen confirmación que la probabilidad de colisión absoluta de un paquete cualquiera. De cara al usuario, lo que importa es si sus paquetes llegan o no.

Calcule la probabilidad de colisión en base exclusivamente a los paquetes de llegadas originales. Puede hacerlo en contando los ACKs correctamente recibidos.

⁷ <https://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html>

Imprima por salida de error la intensidad de tráfico ofrecida por las llegadas frente a la intensidad de tráfico de usuario correctamente confirmada.

```
$ java AlohaAckRtx 0.5 0.25 0.05 0.05 10000 243305 1>/dev/null
0.125000 0.076398
```

Simule distintos valores para tiempos de retardo y margen $D = M = 0.05$. Represente estos resultados junto con los obtenidos en la simulación de ALOHA puro sin ACKs ni retransmisiones.

```
$ gnuplot -p -e "plot 'sim_AlohaPuro.txt' using 1:2, 'sim_AlohaAckRtx.txt'
using 1:2, x * exp(-2*x)"
```

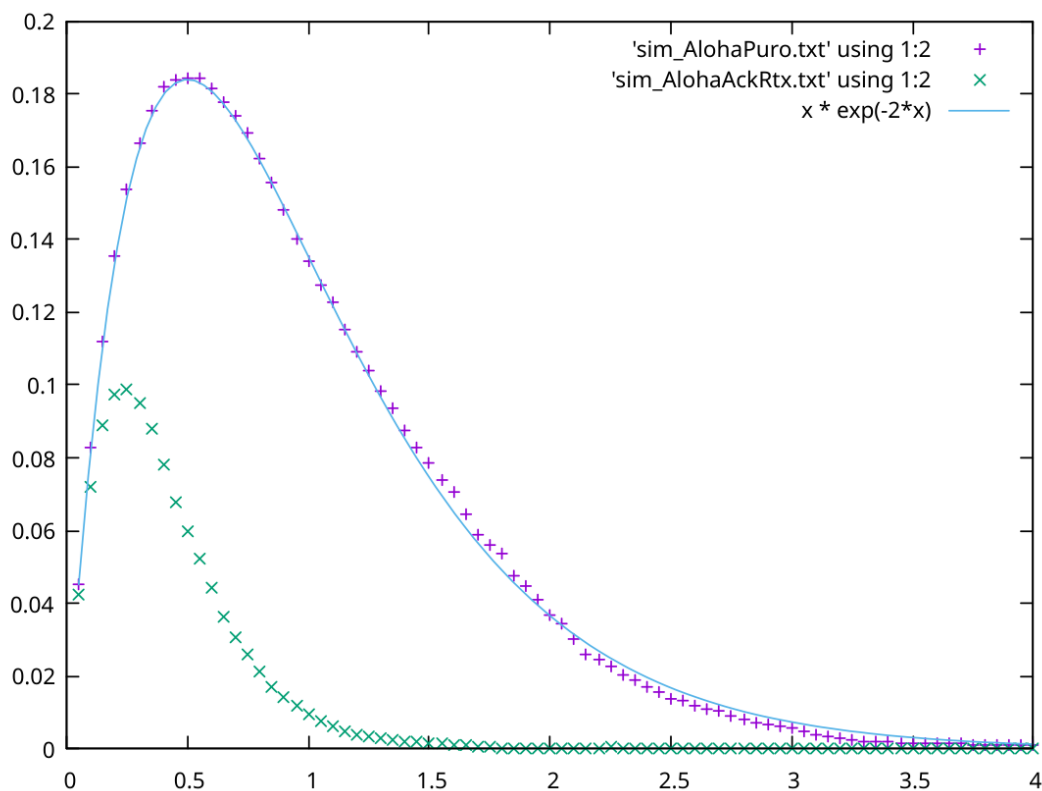


Figura 6 – Tráfico cursado frente al tráfico ofrecido. Comparación ALOHA puro y ALOHA con ACKs y retransmisiones.

7. ALOHA: modelos matemáticos

No se sugiere leerlo, pero si quieren hacerse a la idea de la complejidad que puede tener analizar estos sistemas de forma matemática pueden ojear algo como [1]. En la asignatura no vamos a hacer nada parecido y como mucho en alguna asignatura futura se verá en uno de sus temas un poco sobre la intuición por detrás de este tipo de soluciones. Tradicionalmente es una problemática tratada en nivel de máster y con conocimiento de más herramientas matemáticas generales (transformadas) y estadísticas.

[1] R. T. B. Ma, V. Misra and D. Rubenstein, "An Analysis of Generalized Slotted-Aloha Protocols," in *IEEE/ACM Transactions on Networking*, vol. 17, no. 3, pp. 936-949, June 2009, doi: [10.1109/TNET.2008.925633](https://doi.org/10.1109/TNET.2008.925633).

<https://www.ee.columbia.edu/~tbma/StudyArea/Publications/ToNAloha.pdf>