Redes de ordenadores
Transmisión libre de errores de un fichero

1 Introducción

El propósito de este trabajo es el desarrollo de una aplicación original para la transferencia de un fichero, utilizando como servicio de transporte el protocolo UDP. La aplicación resultante deberá ser capaz de adaptarse a escenarios de red diversos, manteniendo unas buenas prestaciones.

La aplicación consistirá en dos procesos independientes, uno actuando de emisor de la información (fuente), y el otro de receptor (sumidero). Diferentes escenarios de red se emularán con la ayuda de un tercer programa capaz de alterar la secuencia de los paquetes recibidos, de introducir errores en la transmisión, retardos de propagación... Este emulador de red ya está implementado, y podéis descargarlo de `https://github.com/RedesdeOrdenadores/ShuffleRouter` o de `https://snapcraft.io/shufflerouter`.[1]. Para ver las opciones, podéis ejecutar *shufflerouter -h*

2 Los tres procesos

2.1 El emulador de red

Se trata de una sencilla aplicación -que ya se os da hecha- que debe recibir, a modo de paso intermedio obligatorio, todo paquete transmitido por ambos extremos de la comunicación (fuente y sumidero), reenviándolos a su destino natural tras retrasarlos un tiempo aleatorio entre `min_delay` y `min_delay` + `rand_delay`. Como consecuencia del retardo introducido, los paquetes pueden abandonar este proceso fuera de orden.

El emulador de red también puede descartar paquetes con probabilidad `drop`.

El emulador de red debe recibir el identificador de la aplicación destino (dirección IP y puerto), a donde se dirige un paquete que lo atraviesa, en los primeros 6 octetos de la cabecera de aplicación que habrá incorporado el extremo transmisor correspondiente. Los primeros cuatro octetos contendrán la dirección IP destino, mientras que el quinto y sexto, el número de puerto, ambos en formato de red (big-endian). A modo de ejemplo, los primeros seis bytes de un paquete con IP y puerto destino (127.0.0.1, 2000) serían, en notación hexadecimal, 0×7f 0×00 0×00 0×01 0×07 0×d0. El emulador modificará estos seis octetos al reenviar el paquete al destino con la dirección IP original y el puerto de la aplicación de origen.

2.2 La aplicación emisora

Su objetivo es leer un fichero y transmitirlo, de forma fiable y con eficiencia, a la aplicación receptora. Su invocación por línea de comandos requiere cinco argumentos:

```
ro1920send input_file dest_IP dest_port emulator_IP emulator_port
```

Para hacer su trabajo, la aplicación emisora del fichero deberá implementar un algoritmo de retransmisión (ARQ), y necesitaréis definir un protocolo de comunicación para que ambos

[1] Recomendamos instalar la versión basada en snap, ya que se actualizaría automáticamente en caso de que detectemos algún fallo/defecto en su funcionamiento.

extremos se entiendan. Las características del algoritmo ARQ, y la definición de los mensajes que implica su operación, se dejan libres a vuestra elección. Eso sí, tened en cuenta que el emisor no podrá conformar paquetes mayores de 1472 bytes.

Una vez finalice la transmisión (y todos los datos hayan sido asentidos por el receptor), la aplicación emisora debe acabar automáticamente.

2.3 La aplicación receptora

La ejecución de la aplicación receptora del fichero requiere dos argumentos en la línea de comandos:

```
ro1920recv output_file listen_port
```

Este extremo de la comunicación podrá conocer la dirección IP y el puerto donde escucha el emulador de red inspeccionando los metadatos asociados a los datagramas recibidos. No olvidemos que todo paquete, recibido y transmitido, tiene que atravesar el emulador de red.

La aplicación receptora debe finalizar su ejecución tras recibir todo el fichero transmitido. Y una posibilidad para detectar dicho final es la utilización de un paquete de datos de tamaño cero para que la fuente señalice al receptor tal eventualidad.

3 Comprobación de los argumentos de entrada

Ambos programas solo tendrán que comprobar que el número de argumentos de entrada es correcto. En caso de que este chequeo falle, el correspondiente programa finalizará su ejecución, mostrando previamente un mensaje sugiriendo la sintaxis de invocación correcta.

4 Entrega y evaluación

El código fuente podrá desarrollarse en C, C++, Python 3 o Java. En cualquier caso, las versiones requeridas para probar/evaluar el programa tienen que ser las ya instaladas en los laboratorios de la Escuela.

Los dos programas a desarrollar deben llamarse `ro1920send` y `ro1920recv`, respectivamente.

En caso de programar en C o C++, es obligatorio incluir un `Makefile` que automatice la compilación del proyecto. Si el desarrollo es en Java, la clase principal tiene que residir en el paquete por defecto (*default package*).

La fecha límite para la entrega del código, que deberá ser **original** y **meridianamente legible**, es el 8 de Mayo a las 23:55.

Se entregará un único fichero ".zip" que contenga una carpeta donde resida vuestro trabajo. El nombre de dicha carpeta (y, en consecuencia, la versión comprimida a entregar) será el DNI del estudiante (e.g.: 12345678.zip). Las entregas que no sigan este formato no serán admitidas.

Utilizaremos un procedimiento de evaluación que se apoye en las prestaciones del producto, midiendo, para diferentes escenarios, el tiempo invertido y el número de bytes transmitidos para la **correcta** transferencia de un fichero. En la tabla se muestran las pruebas que se realizarán para obtener la calificación indicada en la primera columna. Más adelante se completará la tabla indicando los valores máximos que se deben obtener para el tiempo de transferencia y la cantidad total de bytes transmitidos, para enviar correctamente el fichero. Huelga decir que a mayor sofisticación del algoritmo ARQ implementado, mejor nota.

Mark	File size (MB)	Min. delay (ms)	Rand. delay(ms)	Drop rate	Tiempo (s)	Bytes tx. (MB)
5	10	5	0	0.1%		
+0.5	10	5	0	0.1%		
+0.5	5	20	0	0.1%		
+0.25	5	20	0	0.1%		
+0.25	5	5	15	0.1%		
+0.25	5	5	15	1%		
+0.25	100	5	5	0.1%		
+1	200	50	0	0.01%		
+1	200	100	0	0.1%		
+1	200	0	100	10%		

Cuadro 1: Relación entre prestaciones y calificación final.